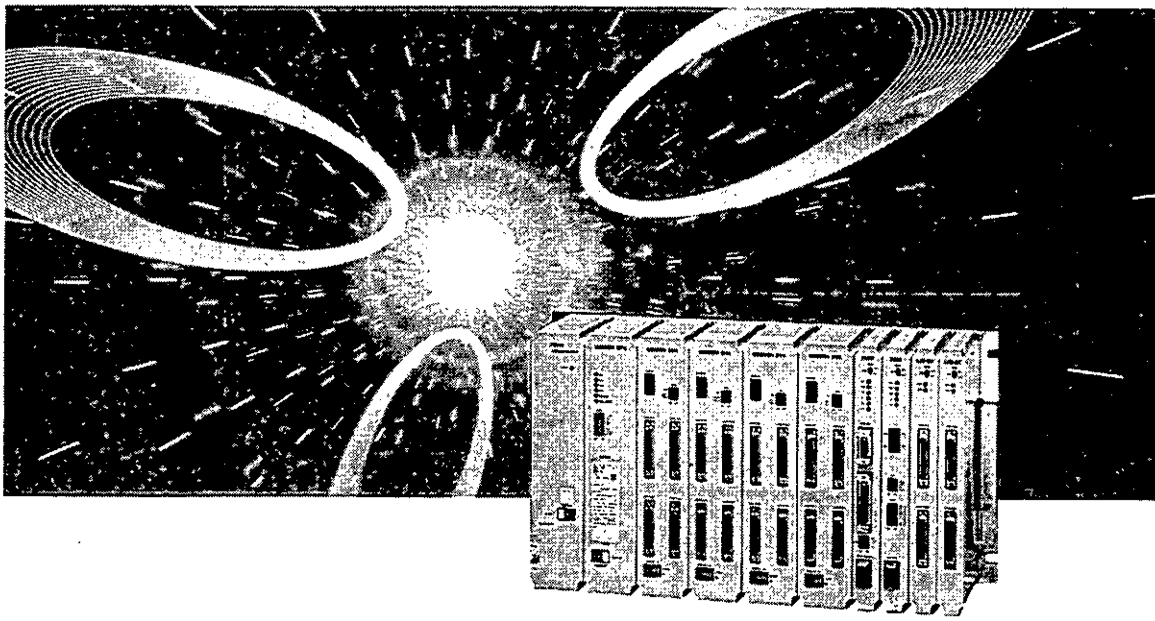
MACHINE CONTROLLER CP-9200SH
PROGRAMMING MANUAL



YASKAWA

MANUAL NO. SIE-C879-40.3B

This Programming Manual provides descriptions on the programming language which is essential for preparing the software for the Machine Controller CP-9200SH.

In this manual, "CP-717" refers to Control Pack CP-717, which is one of the peripheral devices.

Listed below are other documents relevant to the CP-9200SH. Please refer to these materials also.

■ Relevant Documents

Document No.	Name of Document
SIE-C873-16.4	FDS System Installation Manual
SIE-C877-17.4	Control Pack CP-717 Operation Manual (Vol.1)
SIE-C877-17.5	Control Pack CP-717 Operation Manual (Vol.2)
TOE-C877-17.7	Control Pack CP-717 Instructions
CHE-C879-40	CP-9200SH Brochure
KAE-C879-40	CP-9200SH Catalog
SIE-879-40.1	Machine Controller CP-9200SH User's Manual
SIE-879-40.2	Machine Controller CP-9200SH Servo Controller User's Manual

TABLE OF CONTENTS

1 INTRODUCTION TO PROGRAMMING

The programming languages that can be used with CP-9200SH are described in this chapter.

1.1 Programming Languages

CP-9200SH support the programming languages shown in Table 1.1. User programs can be prepared using the programming language that is optimal for the application. For details, refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1).

Table 1.1 Programming Languages that can be Used

Programming Language	Characteristics
Ladder program	• Programs are prepared using relay circuit instructions and text type instructions (control instructions, numerical operation instructions, etc.) • Sequential processes, numerical operation processes,data processes, and various other programs can be written.
Table format program	• Programs for specific applications are prepared in FIF (fill in form) with the use of tables. • Tables, such as the constant data setting table, interlock table, and part composition table, are available.
SFC (sequential function chart) program	• Sequential programs are prepared in flowchart form by the use of steps and transition conditions. • Sequences, such as automatic operation flows, can be written readily.

2 HIERARCHICAL STRUCTURE OF THE DRAWING SYSTEM AND PROGRAMS

Drawings, which are the basic programming units, and their hierarchical structure and function definition methods are described in this chapter.

User programs are managed in units of drawings, which are identified by the drawing No. (DWG No.). These drawings serve as the basis of user programs.
There are parent drawings, child drawings, grandchild drawings, and operation error processing drawings. Besides drawings, there are also functions, which can be referenced freely from each drawing.

Parent Drawings

The parent drawing is executed automatically by the system program when the "Condition of Execution" of Table 2.1 is established.

Child Drawings

Child drawings are executed upon being referenced from the parent drawing by the SEE Instruction.

Grandchild Drawings

Grandchild drawings are executed upon being referenced from a child drawing by the SEE Instruction.

Operation Error Processing Drawing

This is executed automatically by the system program upon occurrence of operation error.

Functions

Functions are executed upon being referenced from the parent, child, or grandchild drawing by the FSTART Instruction.

2.1 Types and Priority Levels of Parent Drawings

Parent drawings are classified by the first character of the drawing (A, I, H, L) according to the purpose of the process. The priority levels and execution conditions of drawings are defined as shown in Table 2.1. For details, refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1).

Table 2.1 Types and Priority Levels of Parent Drawings

Type of Parent Drawing	Role of Drawing	Priority Level	Condition of Execution	Number of Drawings (Note)
DWG. A	Starting process	1	Turning on the power (Executed once when the power is turned on.)	64
DWG. I	Interruption process	2	Start of interruption (Executed upon rising of interruption input signal.)	64
DWG. H	High-speed scan process	3	Start of fixed cycle (Executed on each high-speed scan time.)	100
DWG. L	Low-speed scan process	4	Start of fixed cycle (Executed on each low-speed scan time.)	100

(Note) : The details of the number of drawings is as follows.
Parent drawing : 1 (□)
Operation error processing drawing : 1 (□00)
Child drawings : } n-2 (□01 to 99) } A maximum total of n-2 child
Grand child drawings : (□ΔΔ.01 to 99) and grandchild drawings.
(A, L: 62, H, L: 98)

* n: the maximum number of drawings that can be used.
□: first character of the drawing (A, I, H, L)
ΔΔ: child drawing number

2.2 Execution Control of Parent Drawings

2.2.1 Execution Control of Parent Drawings

Each drawing is executed based on its priority level as shown in Fig. 2.1.

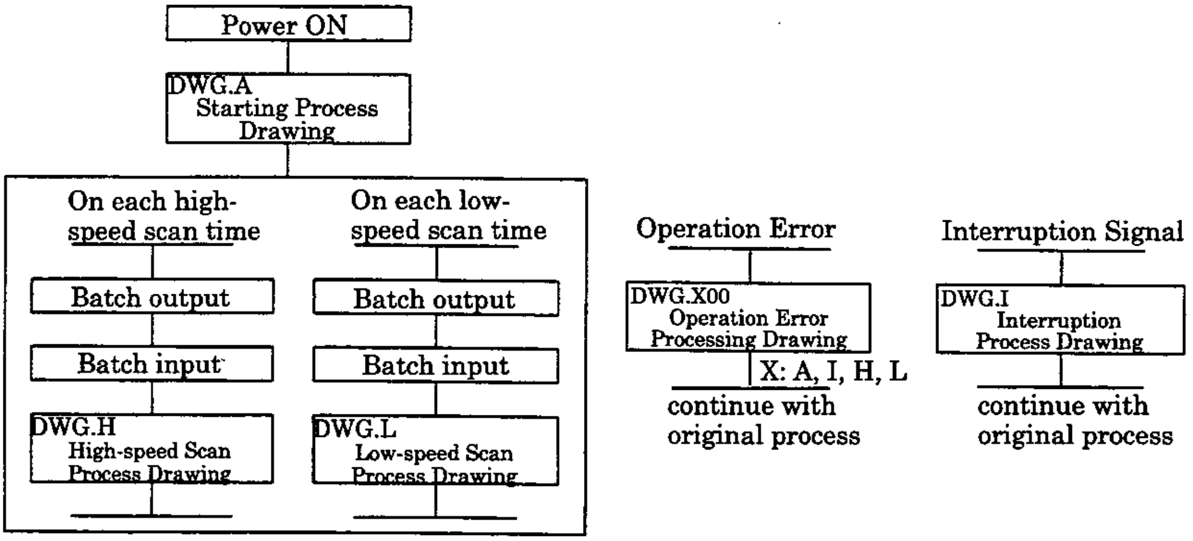
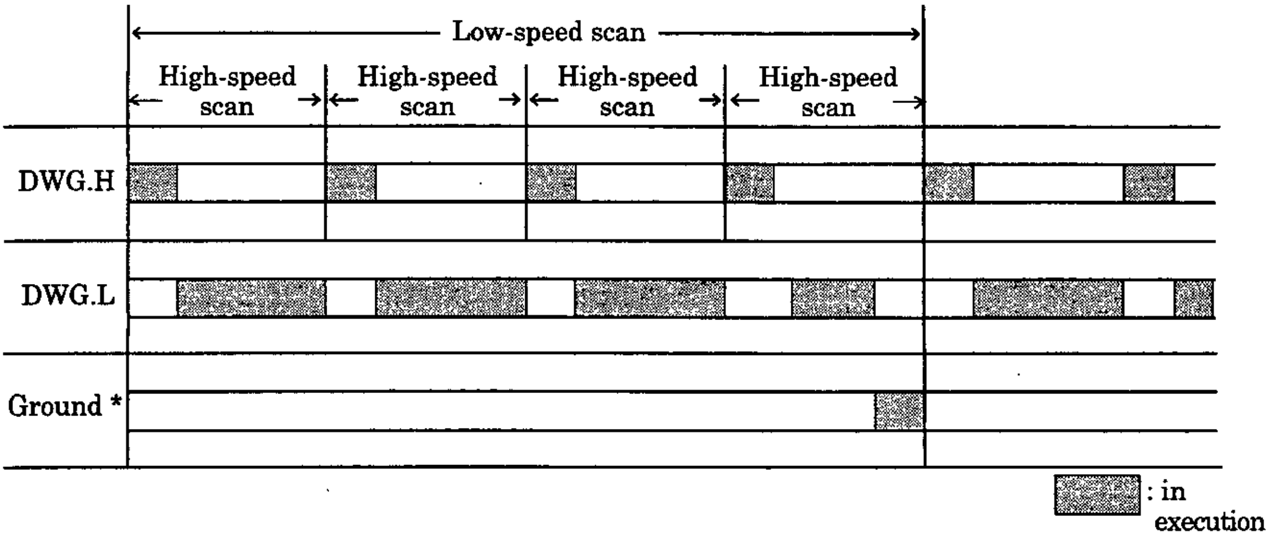

Power ON
DWG.A
Starting Process Drawing
On each high-speed scan time
Batch output
Batch input
DWG.H
High-speed Scan Process Drawing
On each low-speed scan time
Batch output
Batch input
DWG.L
Low-speed Scan Process Drawing
Operation Error
DWG.X00
Operation Error Processing Drawing
X: A, I, H, L
continue with original process
Interruption Signal
DWG.I
Interruption Process Drawing
continue with original process

Fig. 2.1 Execution Control of Parent Drawings

2.2.2 Scheduling of the Execution of Scan Process Drawings

The scan process drawings are not executed simultaneously but are scheduled based on priority levels as shown in Fig. 2.2 and are executed on the schedule.



* : For executing internal processes (self-diagnosis, etc.) of the system.

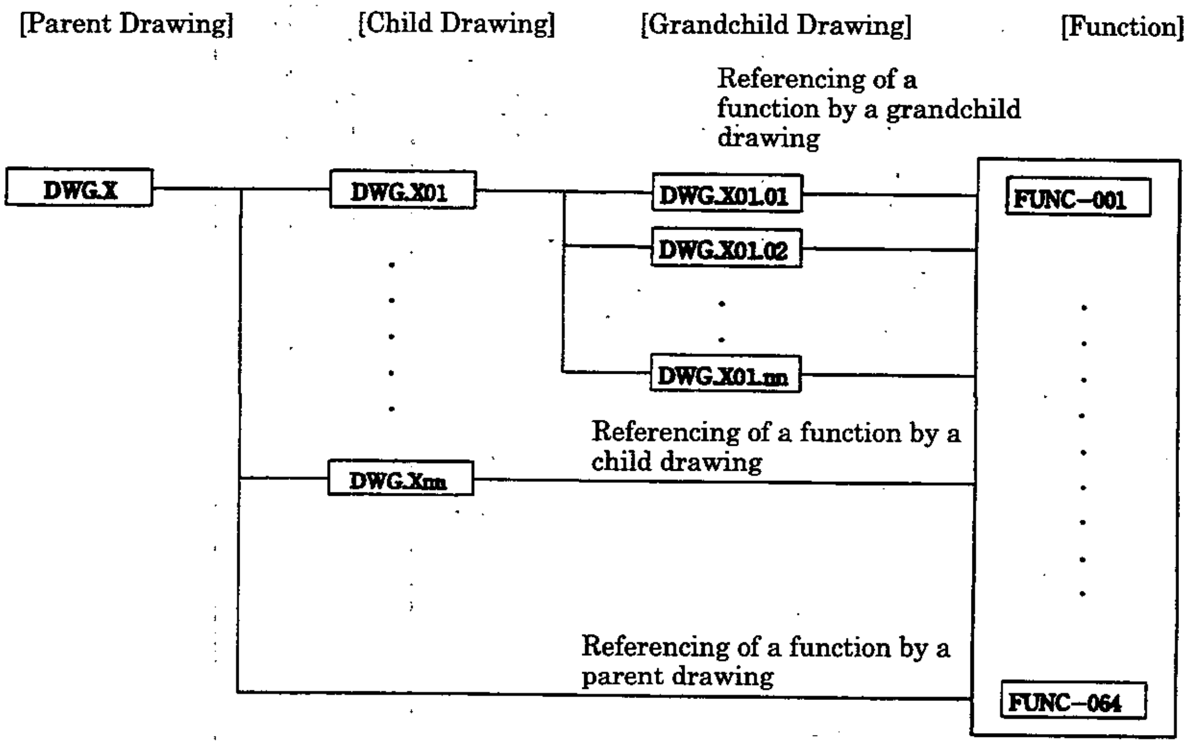
Fig. 2.2 Scheduling of the Execution of Scan Process Drawings

2.3 Hierarchical Structure of Drawings

The drawings are arranged in the manner: parent drawing - child drawing - grandchild drawing. However, a parent drawing cannot reference a child drawing of a different type and a child drawing cannot reference a grandchild drawing of a different type. The child drawing is referenced from the parent drawing, and from that child drawing the grandchild drawing is referenced. This structure is always followed, and is called the hierarchical structure of drawings.

2.3.1 Execution of Drawings

The user prepares each processing program with a parent drawing - child drawing - grandchild drawing hierarchy as shown in Fig. 2.3.


[Parent Drawing]
[Child Drawing]
[Grandchild Drawing]
[Function]
Referencing of a function by a grandchild drawing
DWG.X
DWG.X01
DWG.X01.01
DWG.X01.02
DWG.X01.nn
FUNC-001
Referencing of a function by a child drawing
DWG.Xnn
Referencing of a function by a parent drawing
FUNC-064

(Note) Substitute A, I, H, or L in X.

Fig. 2.3 Hierarchical Structure of DWG's

The parent drawing is executed automatically by the system, since from Table 2.1 of 2.1 "Types and priority of parent drawings," criteria for execution are determined separately for each type. In other words, the parent drawing is automatically called (called up and executed) by the system. Thus, the customer can execute any child or grandchild drawing by programming a DWG reference instruction (SEE instruction) in the parent or child drawings.
Functions listed in 2.2 may be referenced from all drawings. Furthermore, a function can be referenced by a function.
If a operation error occurs, operation error processing drawings corresponding to each screen will be started.

2.3.2 Execution Process of Drawings

The execution process of the drawings arranged in a hierarchy is carried out in a manner whereby lower-ranking drawings are referenced by upper-ranking drawings.
Taking an example of DWG. A, the hierarchical structure of DWGs (drawings) is shown in Fig. 2.4.

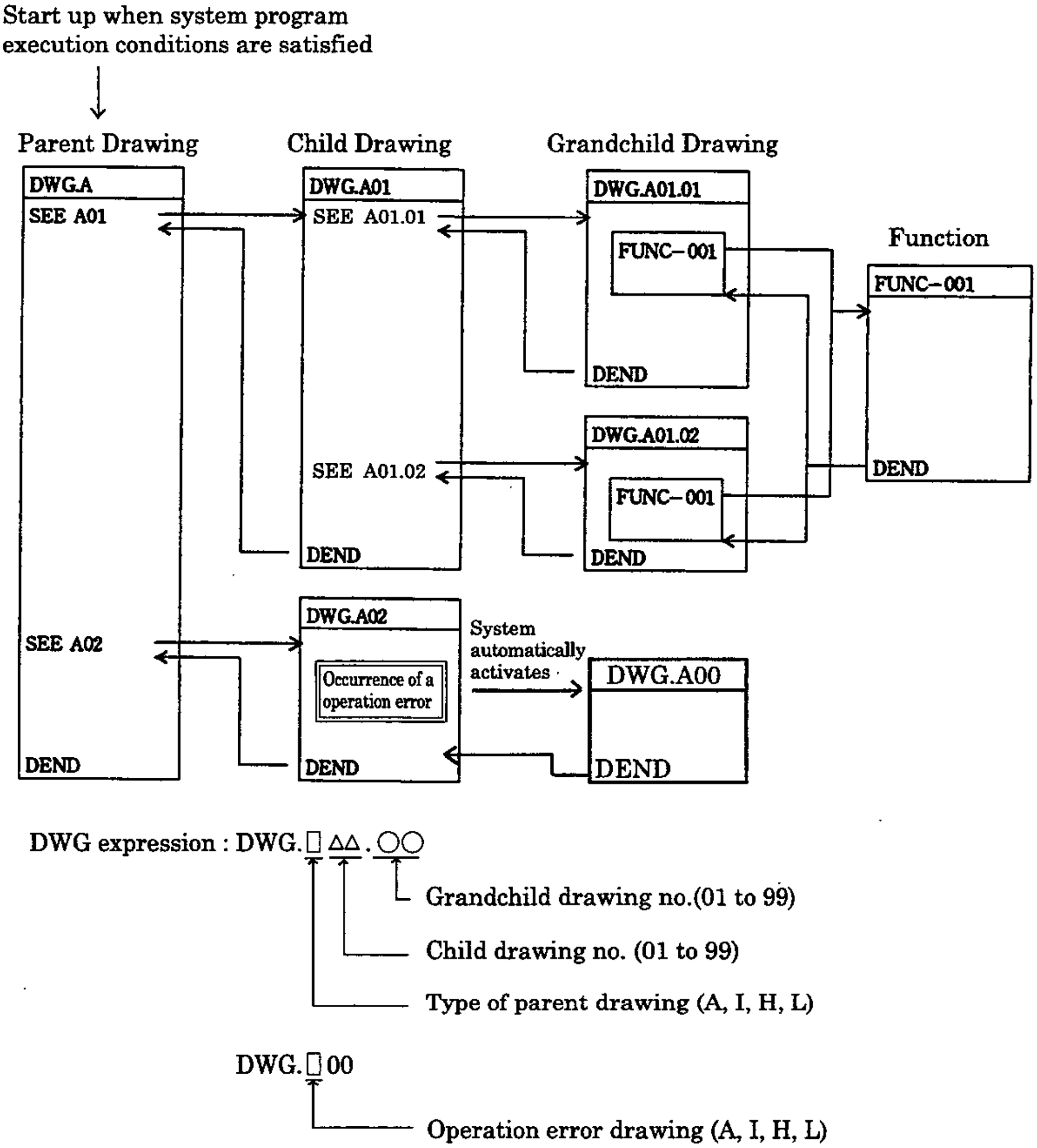

Start up when system program
execution conditions are satisfied
Parent Drawing
Child Drawing
Grandchild Drawing
Function
DWG.A
SEE A01
SEE A02
DEND
DWG.A01
SEE A01.01
SEE A01.02
DEND
DWG.A01.01
FUNC-001
DEND
DWG.A01.02
FUNC-001
DEND
FUNC-001
DEND
DWG.A02
Occurrence of a operation error
DEND
System automatically activates
DWG.A00
DEND
DWG expression : DWG.□ ΔΔ . ○○
Grandchild drawing no.(01 to 99)
Child drawing no. (01 to 99)
Type of parent drawing (A, I, H, L)
DWG.□00
Operation error drawing (A, I, H, L)

Fig. 2.4 Drawing Execution Process

2.4 Functions

Functions can be freely referenced from any drawing. Functions can even be referenced simultaneously from drawings of different types and different hierarchies. Further, functions can also reference other functions. The following benefits can be obtained by using functions.

- It become easy to arrange a program into parts.
- The program can be prepared and maintained easily.

A function is composed of the function definition, which determines the number and types of data that are input into and output from a function, and the main body (program), which depicts the processes that are to be executed according to the inputs and outputs. Functions can be classified into standard system functions, which are made available by the system, and user functions, which are defined by the user.

Standard System Functions

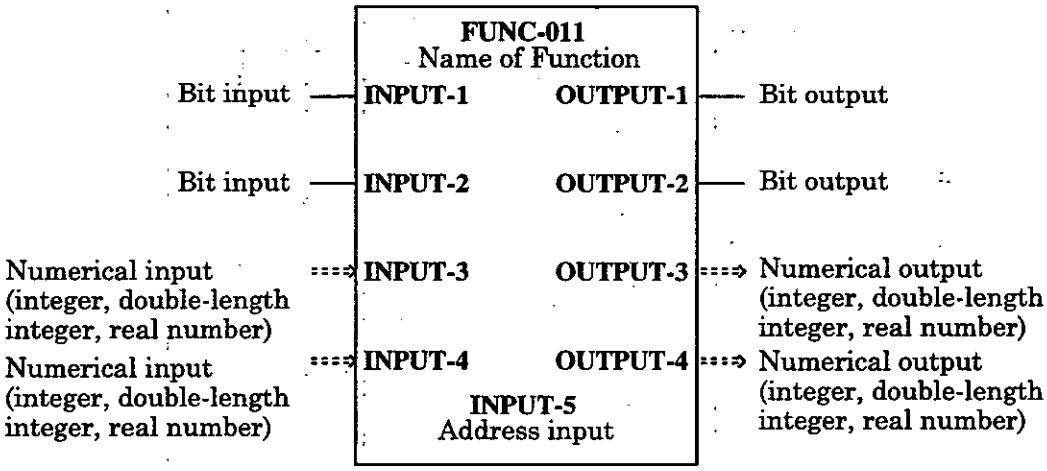
The user can freely use a function that has been predefined by the system, but is not permitted to modify the contents of that function. In other words, the user cannot freely create definitions (program). Refer to Chapter 7 "Standard System Functions" for more information on system functions.

User Functions

These are functions that are defined (programmed) freely by the user. The user prepares the function definition and the main body (program) of the function. See 2.4.2 "User Function Preparation Procedures" concerning the preparation methods.

2.4.1 Function Definition

Functions are defined by the user at the time of user function preparation using the graphic expression form for functions shown in Fig. 2.5.

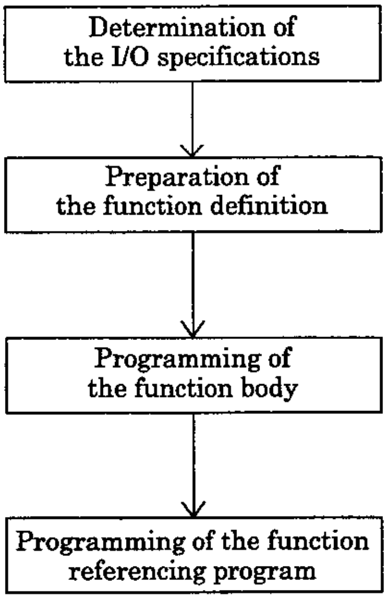

FUNC-011
Name of Function
Bit input
INPUT-1
OUTPUT-1
Bit output
Bit input
INPUT-2
OUTPUT-2
Bit output
Numerical input (integer, double-length integer, real number)
INPUT-3
OUTPUT-3
Numerical output (integer, double-length integer, real number)
Numerical input (integer, double-length integer, real number)
INPUT-4
OUTPUT-4
Numerical output (integer, double-length integer, real number)
INPUT-5
Address input

(Note): The names of the function, the inputs, and the outputs are respectively expressed in 8 or less alphanumeric characters.

Fig. 2.5 Graphic Expression of a Function

2.4.2 User Function Preparation Procedure

Fig. 2.6 shows the procedure for preparing user functions, which can be defined freely by the user.


Determination of the I/O specifications
Preparation of the function definition
Programming of the function body
Programming of the function referencing program

Determine the number of I/Os and the data types.

Input is made using the CP-717.

Prepare in the same manner as the DWGs. However, the types of registers used will differ from those used with the DWG's. Be careful of the correspondence of the register numbers used in the function program and the data input/output upon referencing the function.

Input in the following procedures:

A Input the name of the function with the FSTART Instruction.

B Use the FIN Instruction to prepare the program for input data.

C Use the FOUT Instruction to prepare the program for output data.

* : If a system function is to be used, prepare the program upon referring to the description on I/O definition in Chapter 7 "STANDARD SYSTEM FUNCTIONS". Since the I/O specifications, the function definition, and the main body of the function program are already provided by the system in the case of system functions, these do not have to be defined or prepared.

Fig. 2.6 User Function Preparation Procedure

For more details on operating the CP-717, refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5).

3 REGISTER MANAGEMENT METHOD

Various types of registers are introduced according to application and the register attributes and designation methods are described in this chapter.

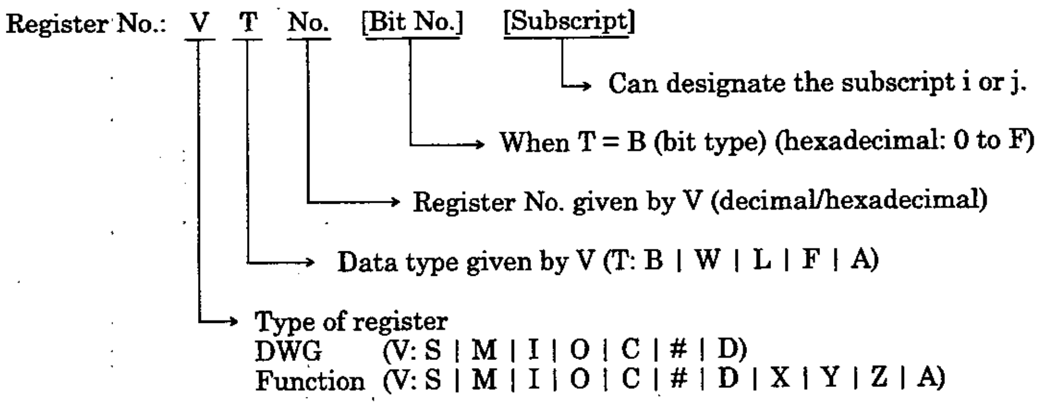
3.1 Register Designation Method

As shown in Table 3.1, registers may be designated by direct register No. designation or by symbolic designation.
These two types of register designation methods may be used together in the user programs. When symbolic designation is to be used, the relationship between the symbol and the register No. must be defined in the symbol table described later.
Refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details.

Table 3.1 Register Designation Methods

Type of Designation	Designation Method	
Direct register No. designation	Bit type register designation Integer type register designation Double-length integer type register designation Real number type register designation Address type register designation	: MB00100A□ : MW00100□ : ML00100□ : MF00100□ : MA00100□
	□ : In the case of subscript designation, the subscript i or j is attached after the register No.	
Symbolic designation	Bit type register designation Integer type register designation Double-length integer type register designation Real number type register designation Address type register designation	: RESET1-A.□ : STIME-H.□ : POS-REF.□ : IN-DEF.□ : PID-DATA.□ An alphanumeric expression of 8 characters or less.
	□ : In the case of subscript designation, a "." and then the subscript, i or j, are attached after the alphanumeric expression of the symbol with 8 characters or less.	

Direct Register No. Designation



Register No.: V T No. [Bit No.] [Subscript]

Can designate the subscript i or j.

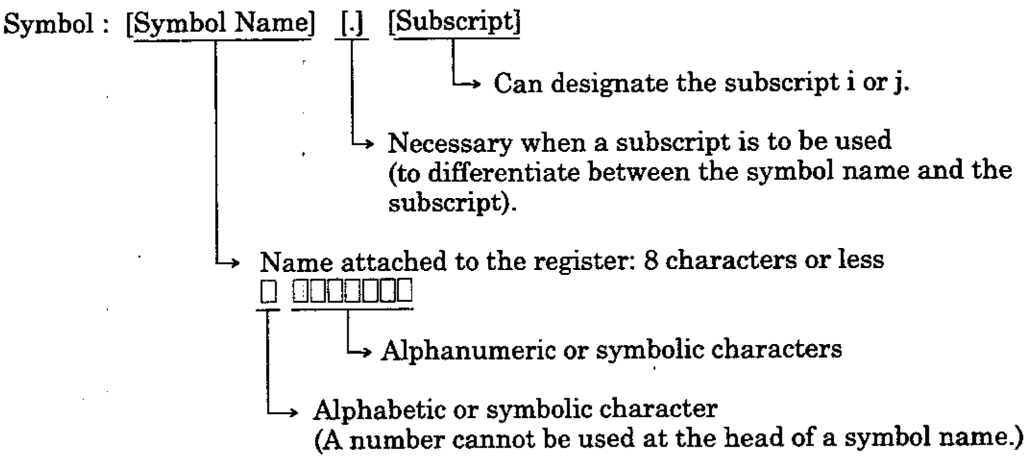
When T = B (bit type) (hexadecimal: 0 to F)

Register No. given by V (decimal/hexadecimal)

Data type given by V (T: B | W | L | F | A)

Type of register
DWG (V: S | M | I | O | C | # | D)
Function (V: S | M | I | O | C | # | D | X | Y | Z | A)

Symbolic Designation



Symbol : [Symbol Name] [.] [Subscript]

Can designate the subscript i or j.

Necessary when a subscript is to be used (to differentiate between the symbol name and the subscript).

Name attached to the register: 8 characters or less
□ □□□□□□□

Alphanumeric or symbolic characters

Alphabetic or symbolic character
(A number cannot be used at the head of a symbol name.)

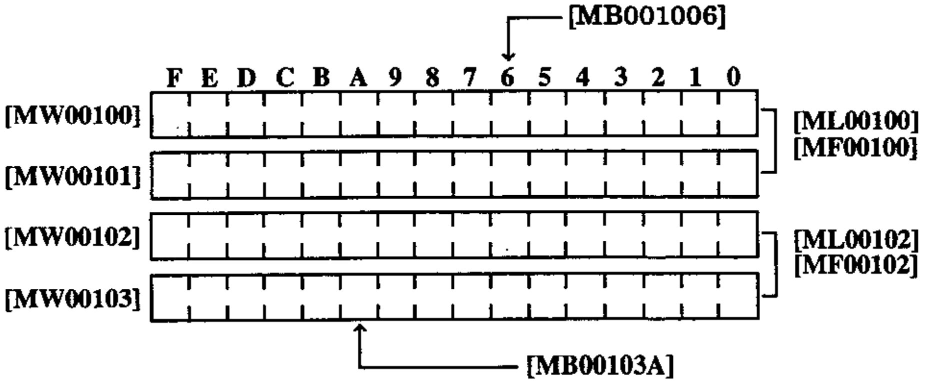
3.2 Data Types

There are five data types; the bit type, the integer type, the double-length integer type, the real number type, and the address type. These are used according to the purpose.
Address type data may be used only for pointer designation.
Refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for the corresponding device for details.

Table 3.2 Data Types

Type	Data Type	Numerical Range	Remarks
B	Bit	ON,OFF	Used for relay circuits.
W	Integer	-32768 to+32767 (8000H) (7FFFH)	Used for numerical operations. Values in ( ) are used in the case of logic operations. Ordinarily used in a series of instruction groups that begin with an integer type entry instruction (⊢). It can also be used in a series of instruction groups that begin with a real number type entry instruction (⊩).
L	Double-length integer	-2147483648 to+2147483647 (80000000H) (7FFFFFFFH)	Used for numerical operations. Values in ( ) are used in the case of logic operations. Ordinarily used in a series of instruction groups that begin with an integer type entry instruction (⊢). It can also be used in a series of instruction groups that begin with a real number type entry instruction (⊩).
F	Real number	±(1.175E -38 to 3.402E +38),0	Used for numerical operations. May only be used in a series that begins with a real number type entry instruction (⊩). Please keep in mind that it cannot be used in a series of instruction groups that begin with an integer type entry instruction (⊢).
A	Address	0 to 32767	Used only for pointer designation.

Register Designation and Data Types



Pointer Designation

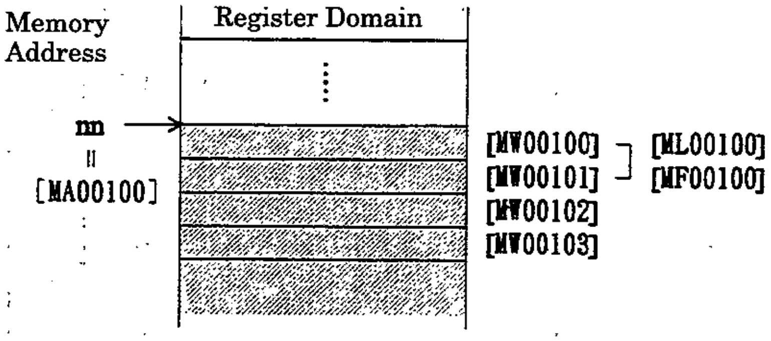


Fig. 3.1 Pointer Designation

In Fig. 3.1, MA00100 signifies the memory address nn of MW00100.
By handing MA00100 to a function, the register domain below MW00100 may be used for internal processes of the function. Such use of an address as an argument of a function is referred to as "pointer designation". In this way, the register domain below MW00100 can be freely used for bits, integers, double-length integers, or real numbers.

3.3 Type of Registers

3.3.1 DWG Registers

The 7 types of register shown in Table 3.3 can be used in each DWG.
Refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details.

Table 3.3 DWG Registers

Type	Name	Designation Method	Description	Characteristic
S	System register	SB, SW, SL, SFnnnnn (SAnnnnn)	Registers made available by the system. The register No. nnnnn is a decimal expression. Upon system start-up, SW00000-SW00049 are all cleared to 0.	Used in common by DWG's
M	Data regoster	MB, MW, ML, MFnnnnn (MAnnnnn)	Registers used in common among DWG's. Used for I/F between DWG's, etc. The register number nnnnn is a decimal expression.	
I	Input register	IB, IW, IL, IFhhhh (IAhhhh)	Register that is used for interface with I/O module and communication module. The register number hhhh is a hexadecimal expression. The register number is assigned on the module configuration definition screen. The register numbers C000 and later are used for interface with motion modules such as SVA modules. For details, refer to the instruction manual of each module.	
O	Output register	OB, OW, OL, OFhhhh (OAhhhh)	Register that is used for interface with I/O module and communication module. The register number hhhh is a hexadecimal expression. The register number is assigned on the module configuration definition screen. The register numbers C000 and later are used for interface with motion modules such as SVA modules. For details, refer to the instruction manual of each module.	
C	Constant register	CB, CW, CL, CFnnnnn (CAnnnnn)	Register that can only be referenced by a program. The register number nnnn is a decimal expression.	
#	# register	#B, #W, #L, #Fnnnnn (#Annnnn)	Registers that can only be referenced in a program. Can only referenced the corresponding DWG. The actual application range is specified by the user with the CP- 717. The register number nnnnn is a decimal expression.	Unique to each DWG
D	D register	DB, DW, DL, DFnnnnn (DAnnnnn)	Internal registers unique to each DWG. Can only referenced the corresponding DWG. The actual application range is specified by the user with the CP- 717. The register number nnnnn is a decimal expression.	

3.3.2 Function Registers

The 11 types of registers shown in Table 3.4 can be used in each function.
Refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details.

Table 3.4 Function Registers

Type	Name	Designation Method	Description	Characteristic
X	Function input register	XB,XW,XL,XFnnnnn	Input into a function Bit input :XB000000 to XB00000F Integer input :XW00001 to XW00016 Double-length integer input: XL00001 to XL00015 The register number nnnnn is a decimal expression.	Unique to each function
Y	Function output register	YB,YW,YL,YFnnnnn	Outputs from a function Bit output :YB000000 to YB00000F Integer output :YW00001 to YW00016 Double-length integer output: YL00001 to YL00015 The register number nnnnn is a decimal expression.	
Z	Register inside function	ZB,ZW,ZL,ZFnnnnn	Internal registers unique to each function. Can be used for internal processes of the function. The register number nnnnn is a decimal expression.	
A	Register outside function	AB,AW,AL,AFnnnnn	External registers that use the address input value as the base address. For linking with (S, M, I, O, #, DAnnnnn). The register number nnnnn is a decimal expression.	
#	# Register	#B,#W,#L,#Fnnnnn (#Annnnn)	Register that can only be referenced by a program. Can reference only the corresponding function. The actual application range is specified by the user with the CP-717. The register number nnnnn is a decimal expression.	
D	D register	DB,DW,DL,DFnnnnn (DAnnnnn)	Characteristic internal register for each function. Can reference only the corresponding function. The actual application range is specified by the user with the CP-717. The register number nnnnn is a decimal expression.	
S	System register	SB,SW,SL,SFnnnnn (SAnnnnn)	Same as the DWG registers. (Since these registers are used in common by both DWG's and functions, be careful of their use when the same function is referenced from DWG's of different priority levels.)	Used in common by DWG's
M	Data register	MB,MW,ML,MFnnnnn (MAnnnnn)		
I	Input register	IB,IW,IL,IFhhhh (IAhhhh)		
O	Output register	OB,OW,OL,OFhhhh (OAhhhh)		
C	Constant register	CB,CW,CL,CFnnnnn (CAnnnnn)		

(Note) SA, MA, IA, OA, DA, #A, and CA may also be used inside a function.

3.3.3 CPU Internal Registers

The registers shown in Table 3.5 are provided inside the CPU. These are used for carrying out user program processes.

Table 3.5 CPU Internal Registers

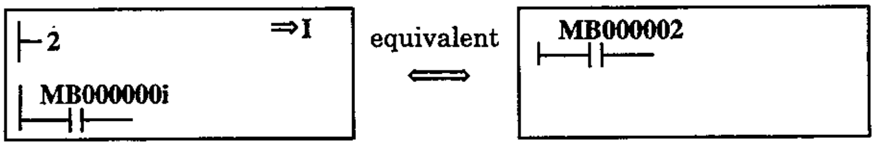
Register	Usage
A register	Used as a register for logic, integer, and double-length integer operations.
F register	Used as a register for real number operations.
B register	Used for relay circuit operations
I register	Used as an index register (I).
J register	Used as an index register (J).

3.3.4 Subscripts i and j

Two types of registers, i and j, are used exclusively for modifying a relay number or register number. i and j have the same function.
These subscripts are explained below with an example for each register data type.

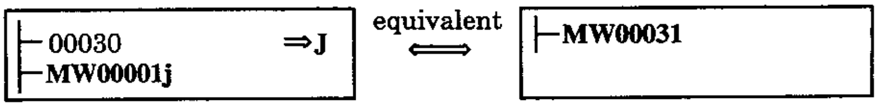
(1) When a Subscript is Attached to Bit Type Data

This will be equivalent to adding the value of i or j to the relay number. For example if I=2, MB000000i will be the same as MB000002. If J=27, MB000000j will be the same as MB000001B.



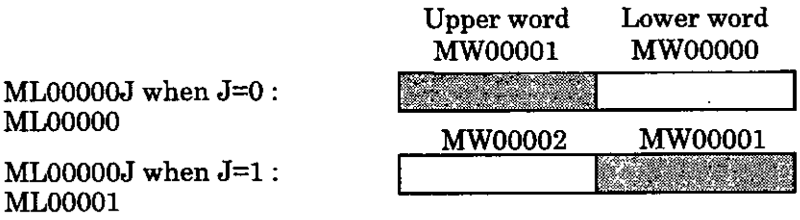
(2) When a Subscript is Attached to Integer Type Data

This will be equivalent to adding the value of i or j to the register number. For example, if I=3, MW00010i will be the same as MW00013. If J=30, MW00001j will be the same as MW00031.



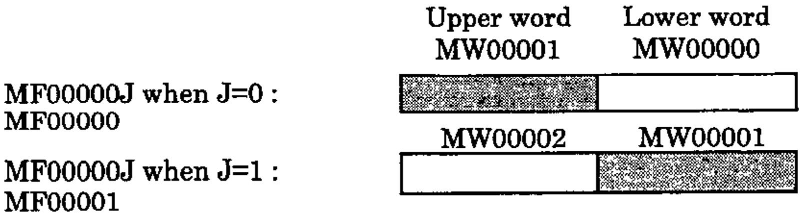
(3) When a Subscript is Attached to Double-Length Integer Type Data

This will be equivalent to adding the value of i or j to the register number. For example, if I=1, ML00000i will be the same as ML00001. ML00000j will be as follows when J=0 and J=1. Be careful.



(4) When a Subscript is Attached to Real Number Type Data

This will be equivalent to adding the value of i or j to the register number. For example, if I=1, MF00000i will be the same as MF00001. MF00000j will be as follows when J=0 and J=1. Be careful



(5) Example of Program Using a Subscript

The program shown in Fig. 3.2 is one in which the total for 100 registers from MW00100 to MW00199 is set in MW00200 by the use of subscript j.

```
├ 00000                                ⟹ MW00200
FOR  J  =00000 to 00099 by 00001
├ MW00200+MW00100j                     ⟹ MW00200
FEND
```

Fig. 3.2 Example of Program Using a Subscript

3.3.5 Function I/O and Function Registers

The inputs and outputs in a function referencing process correspond to the function registers as shown in Table 3.6. Refer to the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details.

Table 3.6 Correspondence between Function I/O's and Function Registers

Function I/O	Function Register
Bit input	The bit number increases continuously from XB000000 in the order of bit input. (XB000000, XB000001, XB000002, ... , XB00000F)
Integer, double-length integer, and real number inputs	The register number increases continuously from XW00001, XL00001, and XF00001 in the order of the integer-double-length integer-real number input. (XW00001, XW00002, XW00003, ... , XW00016) (XL00001, XL00003, XL00005, ... , XL00015) (XF00001, XF00003, XF00005, ... , XF00015)
Address input	The address input value corresponds to register No. 0 of the external register. (Input value = MA00100 : MW00100 = AW00000, MW00101 = AW00001...)
Bit output	The bit number increases continuously from XB000000 in the order of bit output. (YB000000, YB000001, YB000002, ... , YB00000F)
Integer, double-length integer, and real number outputs	The register number increases continuously from YW00001, YL00001, and YF00001 in the order of the integer, double-length integer, and real number output, respectively. (YW00001, YW00002, YW00003, ... , YW00016) (YL00001, YL00003, YL00005, ... , YL00015) (YF00001, YF00003, YF00005, ... , YF00015)

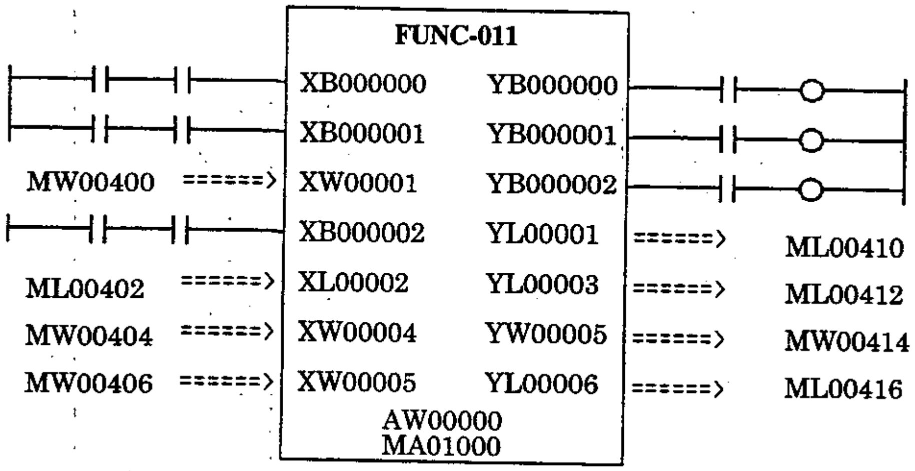
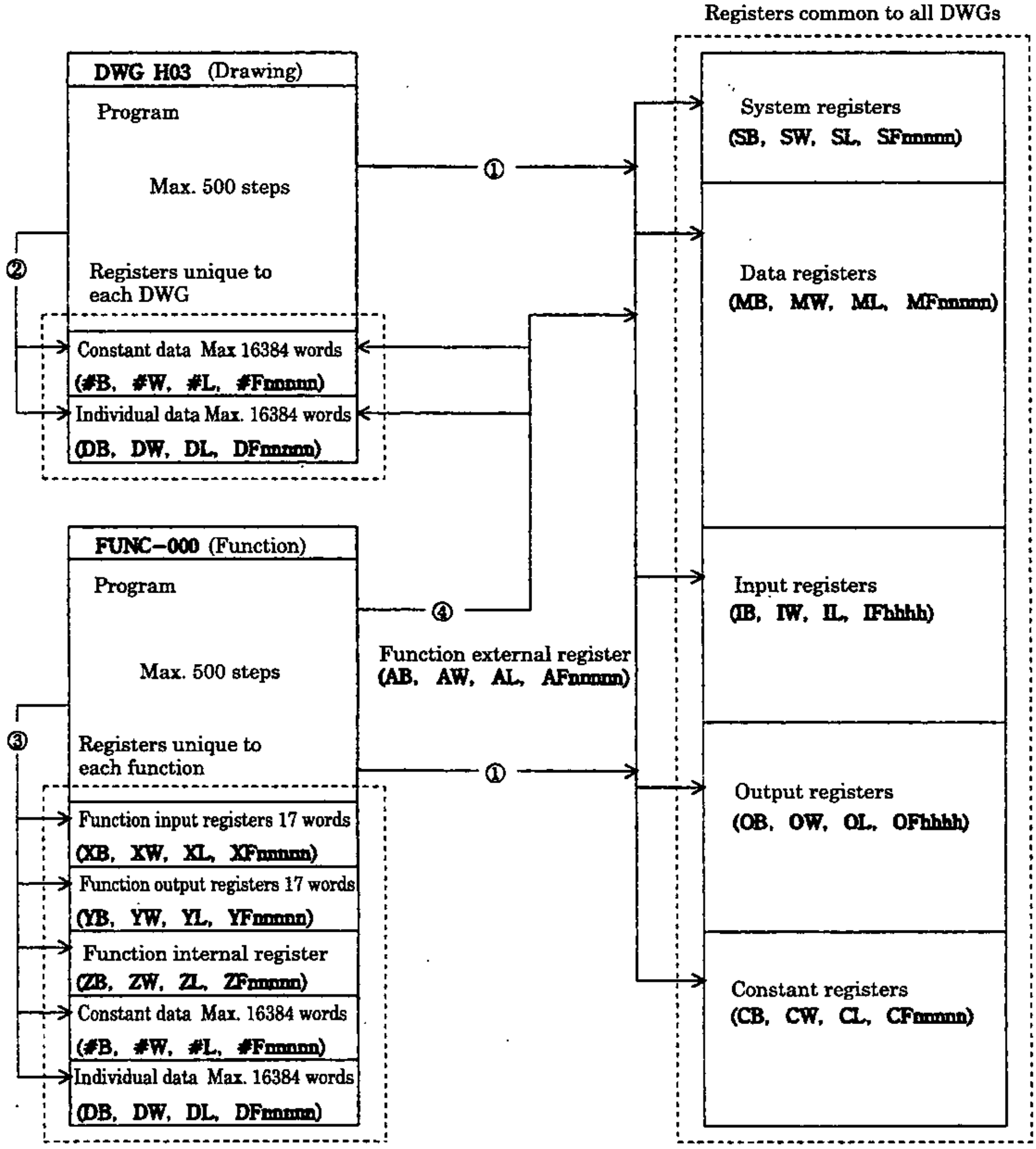


Fig. 3.3 Function Program

In the function program shown in Fig. 3.3, if
"⊢ AW00000 + AW00001 ⇒ AW00002" is written in the program inside the function, the operation:
"⊢ MW01000 + MW01001 ⇒ MW01002" is executed.

3.3.6 Programs and Register Referencing Ranges



① : The registers that can be used in common by the DWG's may be referenced from any drawing or function.
② : Registers that are unique to each drawing can only be referenced within that drawing.
③ : Registers that are unique to each function can only be referenced within that function.
④ : The registers that can be used in common by the DWG's and the registers that are unique to each drawing may be referenced from a function by the use of the function external registers.

3.4 Symbol Management

3.4.1 Symbol Management in the DWG's

All symbols used in the DWG are managed by the DWG symbol table shown in Fig. 3.7. Both registration of symbols on the symbol table and designation of register numbers can be performed on the symbol definition screen of the CP-717. Further, registration, deletion, and modification of symbols as well as designation or modification of register numbers can be done any time while a program is being prepared. A maximum of 200 symbols can be registered for a single drawing. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for the method of defining DWG symbol tables.

- **When an unregistered symbol is used during program preparation...**
  Since only the symbol will be registered automatically in the DWG symbol table, the designation of the register number will become necessary after the program is prepared.

Table 3.7 DWG Symbol Table

No.	Register No.	Symbol	Size *	Remarks
0	IB00000	STARTPBL	1	The register number is a hexadecimal expression.
1	OB00000	STARTCOM	1	The register number is a hexadecimal expression.
2	MW00000	SPDMAS	1	
3	MB000010	WORK-DB	16	
4	MW00010	PIDDATA	10	
5	MW00020	LAUIN	1	
6	MW00021	LAUOUT	1	
:				
N				

* : If a program is prepared using such data configurations as arrays, index process data, etc., define the sizes used in the respective data configurations.
For example, if data is referenced as PIDDATA.i and i takes on values in the range 0 to 9, define the size as 10.

3.4.2 Symbol Management in the Functions

The symbols used in the functions are all managed with the symbol table, shown in Table 3.8. The registration, deletion, and modification of a symbol and the designation and modification of a register number are carried out in the same manner as in the DWG's.

Table 3.8 Function Symbol Table

No.	Register No.	Symbol	Size *	Remarks
0	XB000000	EXECOM	1	
1	XW00001	INPUT	1	
2	AW00001	P-GAIN	1	
3	AB00000F	ERROR	1	
4	YB000000	PIDEXE	1	
5	YW00001	PIDOUT	1	
6	ZB000000	WORKCOIL	4	
7	ZW00001	WORK1	1	
8	ZW00002	WORK2	1	
:				
N				

* : If a program is prepared using such data configurations as arrays, index process data, etc., define the sizes used in the respective data configurations.
For example, if data is referenced as PIDDATA.i and i takes on values in the range 0 to 9, define the size as 10.

3.5 Upward Linking of Symbols and Automatic Number Allocation

3.5.1 Upward Linking of Symbols

The upward linking of symbols refers to the defining of symbols so that symbol names defined in drawings of different hierarchical rank can be used to reference the same register number. Ordinarily, a symbol that is defined for a certain DWG or function becomes unique to that DWG or function program and cannot be referenced by other DWG's or functions.
However, by using the upward linking function for symbols, a symbol defined in a parent drawing may be referenced by a child drawing as long as the drawings are process drawings of the same type. The upward linking of a symbol is set at the Symbol Definition screen of the CP-717. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for details concerning the setting method.

Table 3.9 Linkable Symbols and Symbol Table for Linking

Symbol Table Symbol	Parent drawing	Child drawing	Grandchild drawing
Symbols of a parent drawing	×	×	×
Symbols of a child drawing	○	×	×
Symbols of a grandchild drawing	○	○	×
Symbols inside a function	×	×	×

○: Linkable　×: Not linkable

3.5.2 Automatic Register Number Allocation

Automatic register number allocation refers to the setting of the head register number and the automatic allocation of register numbers to symbols for which register numbers have not been assigned.
Setting automatic allocation of register numbers can be performed on the symbol definition screen of the CP-717. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for detailed procedures for setting them.

Table 3.10 Automatic Register Number Assignment

DWG Symbol Table	Automatic Number Allocation	Function Symbol Table	Automatic Number Allocation
System register S	○	System register S	○
Input regiter I	○	Input register I	○
Output register O	○	Output register O	○
Data register M	○	Data register M	○
# register #	○	# register #	○
C register C	○	C register C	○
D register D	○	D register D	○
————	—	Function input register X	×
————	—	Function output register Y	×
————	—	Function internal register Z	○
————	—	Function external register A	×

○: Automatic number allocation possible
×: Automatic number allocation impossible

4 BASIC INSTRUCTIONS

All of the instructions that can be used with CP-9200SH are described in detail in this chapter.

Arrangement of This Chapter

In this chapter, the description of each instruction is arranged in the following manner.

[Format] Description of the operands and the form of the operands of the instruction.

[Description] Description of the functions of the instruction.

[Operation of the Register]

Shows the storage status of the CPU internal registers.
The registers shown in Table 4.1 are provided inside the CPU. These are used to perform user program processes.

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

A: A register, F: F register, B: B register, I : I register, J : J register

Table 4.1 CPU Internal Registers

Register	Usage
A register	Used as a register for logic, integer, and double-length integer operations.
F register	Used as a register for real number operations.
B register	Used for relay circuit operations
I register	Used as an index register (I).
J register	Used as an index register (J).

[Example(s)] Describes an example or examples of a simple program that uses the instruction.

4.1 Instruction with [ ]

[Format] [Instruction]

[Description] A instruction with [ ] enables conditional execution according to the value of the immediately preceding B register.
The instruction within [ ] is only executed when the value of the B register is ON. [ ] can only be used for 1 instruction. A plurality of instructions cannot be enclosed in a single [ ]. If [ ] is to be used for a plurality of instructions, attach [ ] to each instruction.

[Operation of the Register]

When the B register is OFF:

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

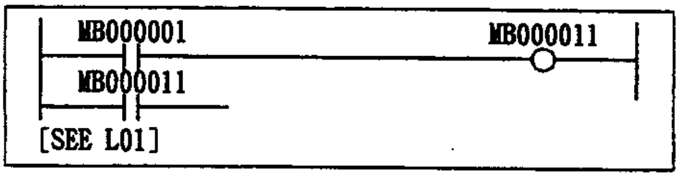
When the B register is ON:

A	F	B	I	J
*	*	*	*	*

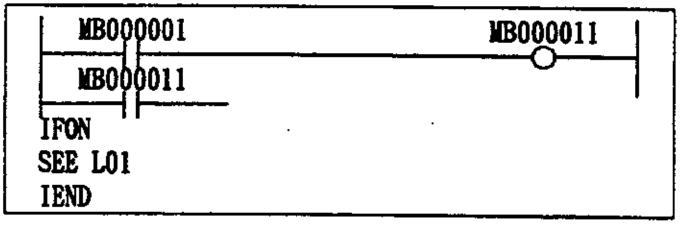
○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

* : In accordance with the instruction within [ ].

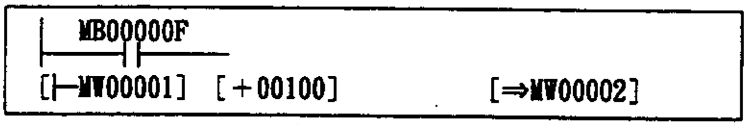
[Example(s)] Example 1



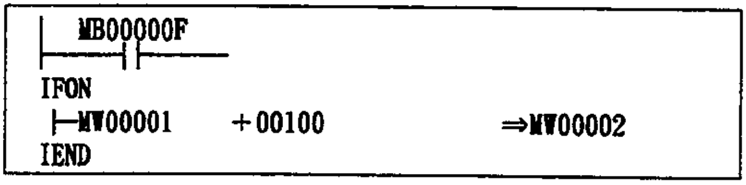
⇕ equivalent



Example 2



⇕ equivalent



4.2 Program Control Instructions

4.2.1 Child Drawing Referencing Instruction (SEE)

[Format] SEE <Child drawing No..or grand-child drawing No.>

[Description] The SEE instruction is used when referencing a child drawing from a parent drawing or when referencing a grandchild drawing from a child drawing. Referencing cannot be performed between drawings which differ in type. For example, "SEE H01" cannot be written inside DWG.L.

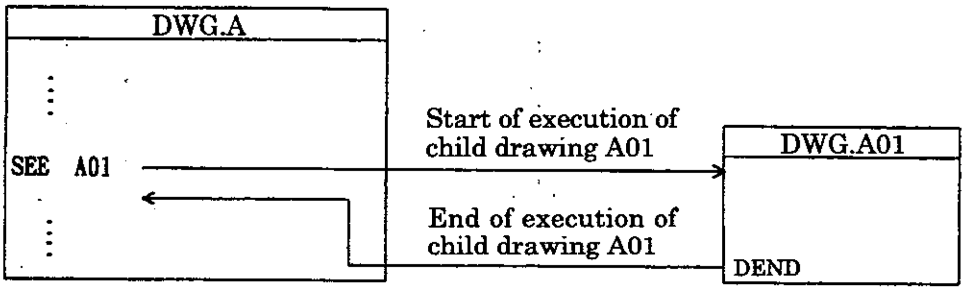
[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]

```
SEE  A01
```


DWG.A
SEE A01
Start of execution of child drawing A01
End of execution of child drawing A01
DWG.A01
DEND

4.2.2 FOR Structure Statement

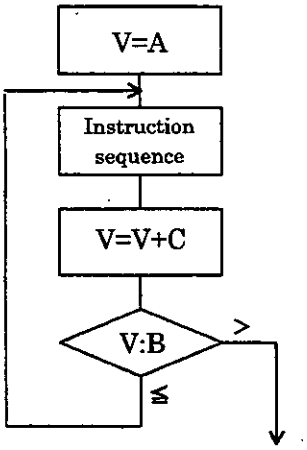
[Format]

```
┌ FOR V = A to B by C
│  Instruction sequence (processing program)
└ FEND
```

[Description] The instruction sequence surrounded by the FOR instruction and the corresponding FEND instruction is repeated by the designated number of times {N = (B - A + 1)/C}. The variable V starts from initial value A and is incremented by C on each repeated execution. The instruction sequence is ended when V>B. The following registers may be used for V, A, B, and C.

V: Any registers of the integer type, any register of the integer type with subscript, and any subscript register (I, J).

A, B, C: Any registers of the integer type, any register of the integer type with subscript, any constant or any subscript register (I, J).
(B>A>0, C>0)



To the next instruction

Fig. 4.1 Execution Control by the FOR Structure Statement

> **Depth of Structure Statements (Nesting)**
>
> The FOR, WHILE, and IF structure statements may contain other structure statements within themselves. This is called "nesting". A FOR, WHILE, or IF structure statement can each be nested up to 8 times. The maximum depth of a nested structure using FOR, WHILE, and IF statements is thus restricted to 24 nests.

[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The total for 100 registers, from MW00100 to MW00199, is stored in MW00200.

```
  ├00000                               ⟹MW00200
┌ FOR J =00000 to 00099 by 00001
│ ├MW00200+MW00100j
└ FEND                                 ⟹MW00200
```

4.2.3 WHILE Structure Statement

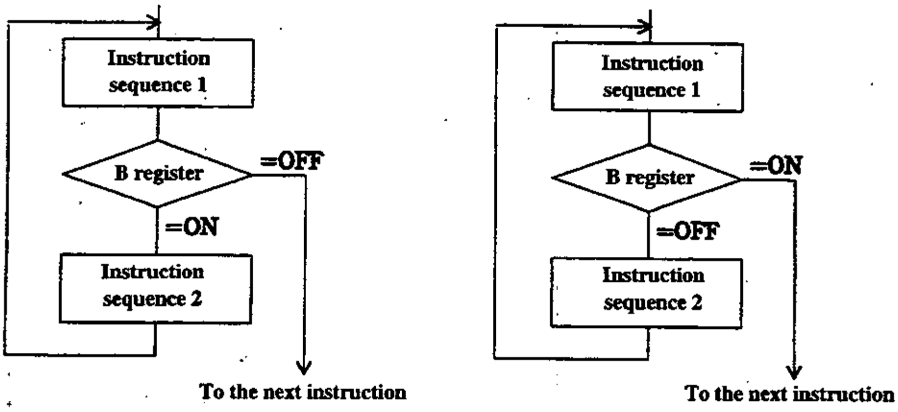
[Format]

```
┌ WHILE
│       Instruction sequence 1 (judgment of repetition condition)
├ ON/OFF
│       Instruction sequence 2 (processing program)
└ WEND
```

[Description] The instruction sequence 2, between WHILE and WEND is executed repeatedly as long as the conditions defined by instruction sequence 1 and the ON (or OFF) instruction are satisfied. When the conditions are no longer satisfied, instruction sequence 2 is not executed and the program proceeds with the instruction next to WEND.

As shown in Fig. 4.2, the condition for execution of instruction sequence 2 is determined by the condition of the B register immediately preceding the ON (or OFF) instruction (ie. the results of instruction sequence 1).

If, for example, the condition for execution is found to be not satisfied as a result of the first execution of instruction sequence 1, the program proceeds with the instruction next to WEND without executing the instruction sequence 2



(a) WHILE-ON-WEND Structure Statement

(b) WHILE-OFF-WEND Structure Statement

Fig. 4.2 Control of Execution by the WHILE Structure Statement

Depth of Structure Statements (Nesting)

The FOR, WHILE, and IF structure statements may contain other structure statements within themselves. This is called "nesting". A FOR, WHILE, or IF structure statement can each be nested up to 8 times. The maximum depth of a nested structure using FOR, WHILE, and IF statements is thus restricted to 24 nests.

NOTE

Write the program so that the condition part (instruction sequence 1) of the WEND structure statement will definitely be unsatisfied at some point. If the repetition is continued endlessly and the program cannot proceed out of the WHILE structure statement, the watchdog timer will be activated and the CPU will stop.

[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

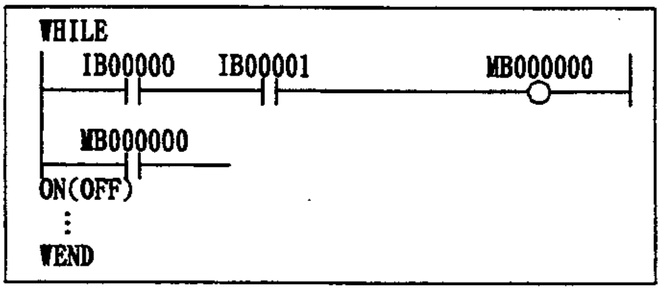
○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The total for 100 registers, from MW00100 to MW00199, is stored in MW00200.

```
├00000                          ⇒I
                                ⇒MW00200
WHILE
├I        <  00100
ON
├MW00200  +  MW00100i           ⇒MW00200
├I        +     00001           ⇒I
WEND
```

NOTE

Place an N.O. contact instruction (⊣⊢) if an ON (or OFF) instruction is to be used after a coil instruction.



4.2.4 IF Structure Statement

The IF structure statement can take one of two formats depending on whether or not an exclusive condition exists. Although the two formats are described separately below, there are no essential differences between these two.

(1) IF Structure Statement - 1

[Format]

```
┌ IFON/IFOFF
│   Instruction sequence (processing program)
└ IEND
```

[Description]

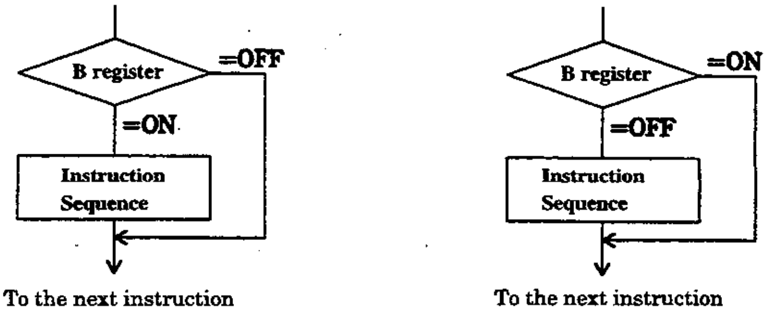
When the IFON Instruction is Used

The instruction sequence between IFON and IEND will be executed if the current value of the B register is ON and will not be executed if the current value of the B register is OFF.

When the IFOFF Instruction is Used

The instruction sequence between IFON and IEND will be executed if the current value of the B register is OFF and will not be executed if the current value of the B register is ON.

The process flows are shown in Fig. 4.3.



(a) IFON-IEND Structure Statement

(b) IFOFF-IEND Structure Statement

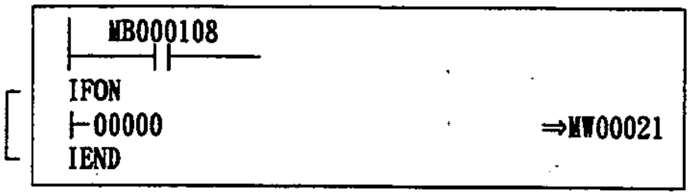
Fig. 4.3 Execution Control by the IF Structure Statement (1)

[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] If MB000108 is ON, the contents of MW00021 are set to 0.



```
 | MB000108
 |----| |----
┌ IFON
│ |-00000                              ⇒MW00021
└ IEND
```

(2) IF Structure Statement - 2

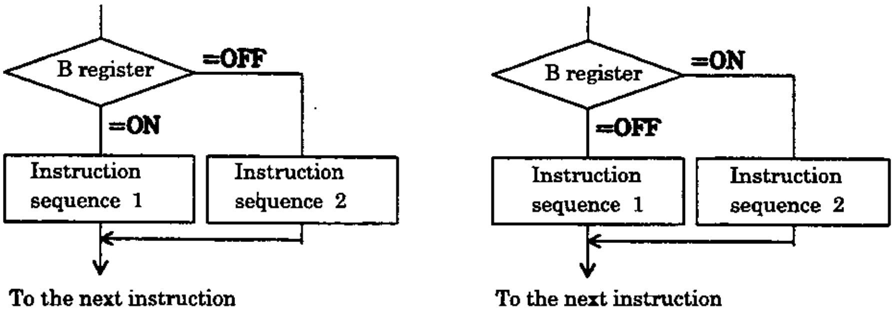
[Format]

```
┌ IFON/IFOFF
│ Instruction sequence - 1
├ ELSE
│ Instruction sequence - 2
└ IEND
```

[Description]

When the IFON Instruction is Used:
If the current value of the B register is ON, only instruction sequence 1 will be executed and instruction sequence 2 will not be executed. If the current value of the B register is OFF, only instruction sequence 2 will be executed and instruction sequence 1 will not be executed.

When the IFOFF Instruction is Used:
If the current value of the B register is OFF, only instruction sequence 1 will be executed and instruction sequence 2 will not be executed. If the current value of the B register is ON, only instruction sequence 2 will be executed and instruction sequence 1 will not be executed.
The process flows are shown in Fig. 4.4.



(a) IFON-ELSE-IEND Structure Statement

(b) IFOFF-ELSE-IEND Structure Statement

Fig. 4.4 Execution Control by the IF Structure Statement (2)

Depth of Structure Statements (Nesting)

The FOR, WHILE, and IF structure statements may contain other structure statements within themselves. This is called "nesting." A FOR, WHILE, or IF structure statement can each be nested up to 8 times. The maximum depth of a nested structure using FOR, WHILE, and IF statements is thus restricted to 24 nests.

[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

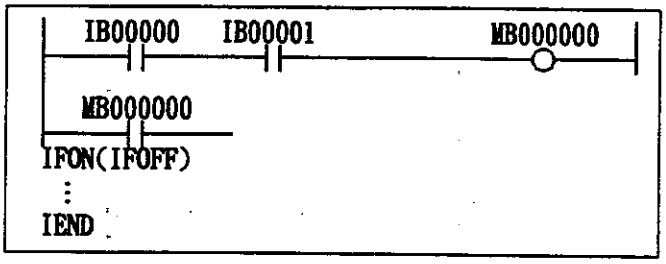
○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]

The contents of MW00011 are set to 0 if MW00010 contains a positive number and to 1 if MW00010 contains a negative number.

```
  ├MW00010 ≧ 00000
┌ IFON .
│ ├00000                        ⇒MW00011
├ ELSE
│ ├00001                        ⇒MW00011
└ IEND
```

NOTE
Place an N.O. contact instruction(┤├)if an IFON (or IFOFF) instruction is to be used after a coil instruction.



4.2.5 Function Referencing Instruction (FSTART)

[Format] FSTART

[Description] The FSTART instruction is used to reference an user function or a system function from a parent drawing, child drawing, or user function. The function definition of the referenced user function must be prepared in advance. System functions do not have to be defined by the user since they are already defined by the system.

[Operation of the Register]

A	F	B	I	J
*	*	*	*	*

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Additional Note]

When "FSTART Enter " is input at the CP-717, the graphic display of the functions is displayed and the input of the function name is prompted. The "FSTART" instruction itself will not be displayed on the screen. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for details on the input method.

4.2.6 Function Input Instruction (FIN)

[Format] FIN

[Description] The FIN instruction is used to store input data into a function input register. The forms of data input into a function register are shown in Table 4.2.

Table 4.2 Function Input Data Forms

Input Data Form	Input Designation*	Description
Bit input	B-VAL	Designates the output to be of a bit type. Usually, the —| |— instruction or the —|/|— instruction is used to reference the function. The bit data become the input to the function.
Integer type input	I-VAL	Designates the input to be of an integer type. Usually, the |— instruction is used to reference the function. The contents (integer data) of the register number designated with the |— instruction become the input to the function.
	I-REG	Designates the input to be the contents of an integer type register. The number of the integer type register is designated when referencing the function. The |— instruction is not necessary. The contents (integer data) of the register with the designated number become the input to the function.
Double-length integer type input	L-VAL	Designates the input to be of a double-length integer type. Usually, the |— instruction is used to reference the function. The contents (double-length integer data) of the register with the number designated with the |— instruction become the input to the function.
	L-REG	Designates the input to be the contents of a double-length integer type register. The number of the double-length integer type register is designated when referencing the function. The |— instruction is not necessary. The contents (double-length integer data) of the register with the designated number become the input to the function.
Real number type input	F-VAL	Designates the input to be of a real number type. Usually, the ||— instruction is used to reference the function. The contents (real number data) of the register with the number designated with the ||— instruction become the input to the function.
	F-REG	Designates the input to be the contents of a real number type register. The number of the real number type register is designated when referencing the function. The ||— instruction is not necessary. The contents (real number data) of the register with the designated number become the input to the function.
Address input	———	Hands over the address of the designated register (an arbitrary integer register) to the function. Only 1 input is allowed in the case of a user function.

* : Indicates the input designation at the CP-717.

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Additional Note]

The graphic display of function inputs is displayed when "FIN Enter " is input at the CP-717 after designating the data. The "FIN" instruction itself will not be displayed on the screen. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for details on the input method.

> **NOTE**
> It is recommended that I-REG, L-REG, or F-REG be used if the I/O data are not of a bit type.

4.2.7 Function Output Instruction (FOUT)

[Format] FOUT

[Description] The FOUT instruction is used to take out the contents of a function output register as output data of the function. The forms of data output from a function are shown in Table 4.3.

Table 4.3 Function Output Data Forms

Output Data Form	Output Designation*	Description
Bit output	B-VAL	Designates the output to be of a bit type. Usually, the —O—| instruction is used to reference the function. The output data (bit data) are stored in the register with the number designated with the —O—| instruction.
Integer type output	I-VAL	Designates the output to be of a Integer type. Usually, the ⇒ instruction is used reference the function. The output data (integer data) are stored in the register with the number designated with the ⇒ instruction.
	I-REG	Designates the output to be the contents of an integer type register. The number of the integer type register is designated when referencing the function. The ⇒ instruction is not necessary. The output data (integer data) are stored in the register with the designated number.
Double-length integer type output	L-VAL	Designates the output to be of a double-length integer type. Usually, the ⇒ instruction is used to reference a function. The output data (double-length integer data) are stored in the register with the number designated with the ⇒ instruction.
	L-REG	Designates the output to be the contents of a double-length integer type register. The number of the double-length integer type register is designated when referencing the function. The ⇒ instruction is not necessary. The output data (double-length data) are stored in the register with the designated number.
Real number type output	F-VAL	Designates the output to be of a real number type. Usually, the ⇒ instruction is used to reference a function. The output data (real number data) are stored in the register with the number designated with the ⇒ instruction.
	F-REG	Designates the output to be the contents of a real number type register. The number of the real number type register is designated when referencing the function. The ⇒ instruction is not necessary. The output data (real number data) are stored in the register with the designated number.

* : Indicates the output designation at the the CP-717.

[Operation of the Register]

	A	F	B	I	J
B-VAL	○	○	×	○	○
I -VAL	×	○	○	○	○
I -REG	○	○	○	○	○
L-VAL	×	○	○	○	○
L-REG	○	○	○	○	○
F-VAL	○	×	○	○	○
F-REG	○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Additional Note]

The graphic display of function outputs is displayed when "FOUT Enter " is input at the CP-717 after designating the data. The "FOUT" instruction itself will not be displayed on the screen. Refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for details.

[Example(s)]

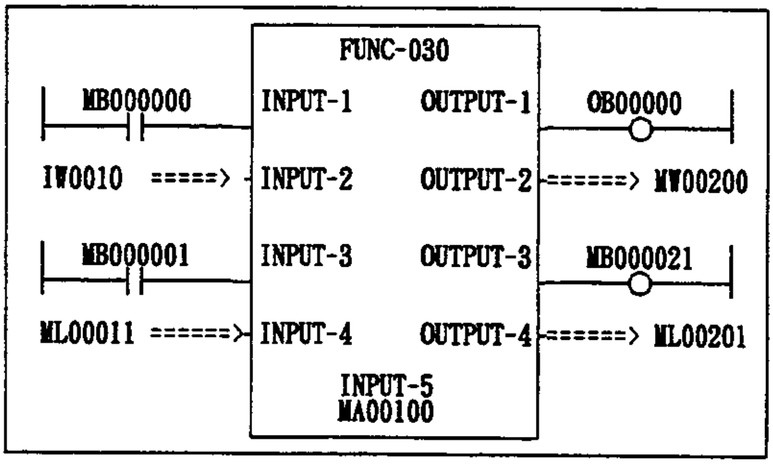


Table 4.4 shows the function I/O data defined by function definition in the program example above.

Table 4.4 Function I/O Data Forms

Input Data	Data Form
INPUT-1	B - VAL
INPUT-2	I - REG
INPUT-3	B - VAL
INPUT-4	L - REG

Output Data	Data Form
OUTPUT-1	B - VAL
OUTPUT-2	I - REG
OUTPUT-3	B - VAL
OUTPUT-4	L - REG

NOTE
It is recommended that I-REG, L-REG, or F-REG be used if the I/O data are not of a bit type.

Table 4.5 shows the correspondence relationships between the I/O data and the function I/O registers when the I/O data are referenced within the main body of the function.

Table 4.5 I/O Correspondence Relationships

Input Data	Referencing within the Main Body of the Function		Output Data
	Function Input Register	Function Output Register	
B register (=MB000000)	XB000000		
IW0010	XW00001		
B Register (=MB000001)	XB000001		
ML00011	XL00002		
MW00100	AW00000		
MW00101	AW00001		
ML00102	AW00002		
MB001040	AB000040		
:	:		
		YB000000	B Register (=OB00000)
		YW00001	MW00200
		YB000001	B Register (=MB000021)
		YW00002	ML00201

4.2.8 Comment Instruction (COMMENT)

Comments can be written at any position in the DWG program or user function program. Alphanumeric characters may be used for comments.

[Format] "character string"

[Description] The character string enclosed with " " is treated as a comment. Since this is merely a comment, it is not executed as an instruction. Be aware that it becomes the target of the number of steps in the user program.
A character string of 12 characters will be equivalent to 1 step (1 basic instruction).

[Operation of the Register]

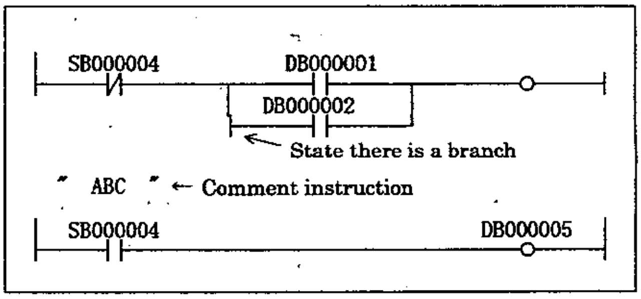
A	F	B	I	J
○	○	○	○	○

○: stored ✕: not stored
* : indeterminate
(Stored or not stored depending on the case.)

NOTE

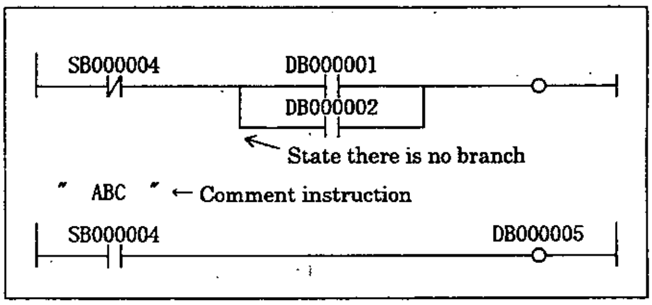
Do not prepare a program that there is a comment instruction in the middle of branching in a series of sequence instruction groups.

<Example 1>



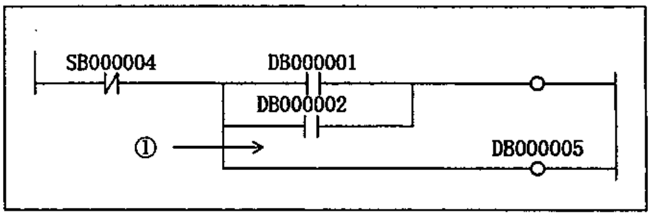
Wrong

↓



Correct

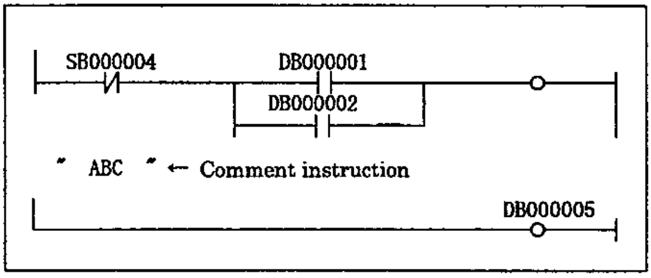
<Example 2>



Correct

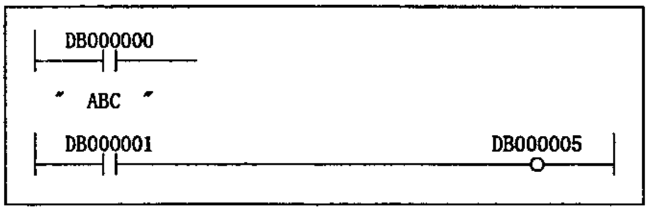
In the diagram above, do not insert a comment instruction at ①.

↓



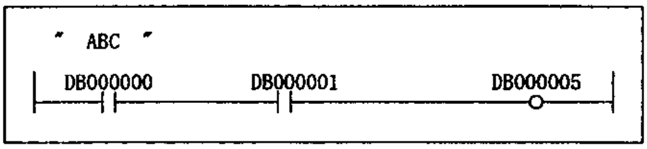
Wrong

<Example 3> Do not prepare a program that there is a comment instruction between contact instructions.



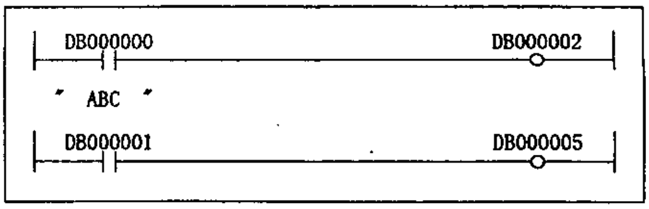
Wrong

↓



Correct

or



Correct

4.2.9 Expansion Program Execution Instruction (XCALL)

[Format] XCALL <type of expansion program>

[Description] The XCALL instruction is used to execute an expasnsion program. Expansion programs refer to the table format programs. There are 4 types of table format programs as shown in Table 4.6. With the CP-9200SH, these expansion programs are converted into ladder programs for execution. A converted ladder program is executed with the XCALL instruction. Although a plurality of XCALL instructions may be used in one drawing, the same expansion program cannot be called more than once.

Table 4.6 Types of Expansion Programs

Symbol	Program Type
MCTBL	Constant table (M register)
IOTBL	I/O conversion table
ILKTBL	Interlock table
ASMTBL	Parts composition table

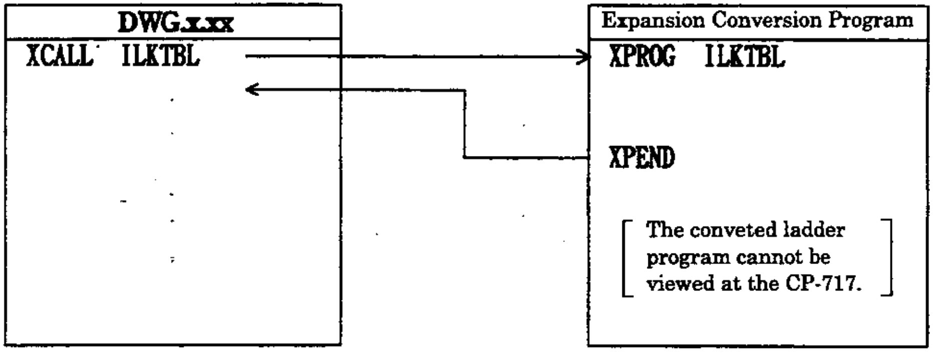
[Operation of the Register]

A	F	B	I	J
○	*	○	*	*

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]

```
XCALL      ILKTBL
```


DWG.xxx
XCALL ILKTBL
Expansion Conversion Program
XPROG ILKTBL
XPEND
The conveted ladder program cannot be viewed at the CP-717.

4.3 Direct I/O Instructions

The direct I/O instructions are used to execute inputs and outputs in an user program independent of the system I/O (batch input/batch output). An input or output is carried out at the point of execution of the direct I/O instruction. The subsequent instruction is not executed until the I/O operation has been completed.

4.3.1 Continuous Execution Type Direct Input Instruction (INS)

[Format] [Parameter/head address of the data table]

INS [Register address (except for #/C)
Register address (except for #/C) with subscript]

[Description] The INS instruction conforming to previously set parameter table contents, continuously performs direct input to a single module. The only modules that can apply direct input are the LIO-01/2000IO. If no error at all occurs, B register is OFF. If an error occurred in even a single word, B register turns ON. During operation, interruption by the system is prohibited.

Table 4.7 INS Instruction Parameter/Data Table

ADR	Type	Symbol	Name	Specifications	Input or Output
0	W	RSSEL	Module designation 1	Designation of module for performing input	IN
1	W	MDSEL	Module designation 2	(The details are described (1) and (2) below.)	IN
2	W	STS	Status	Status for each word output with bit response	OUT
3	W	N	Number of words	Designation of number of continuous input words	IN
4	W	ID1	Input data 1	Outputs the input data.	OUT
⋮	⋮	⋮	⋮	If there is an error, 0 is stored.	⋮
N+3	W	IDN	Input data N		OUT

*** Method of RSSEL and MDSEL Settings**

(1) RSSEL Designates the rack/slot where the target module is mounted.

Hexadecimal expression: xxyyH

xx = rack number ($01_H \leqq xx \leqq 04_H$)

yy = slot number ($00_H \leqq yy \leqq 0D_H$)

However, designate the mounting rack/slot as:

LIO-01: Mounting rack/slot number on LIO-01 itself

2000IO: Mounting rack/slot number on 2000IOIF module connected to the target 2000IO rack

(2) MDSEL

For the LIO-01: Designate the input data offset for the internal LIO-01 module.

For the 2000IO: Designates the rack number/slot number/input module type in the 2000IO rack of the target module.

```
F          CB          87          43          0
|     a     :     b     :     c     :     d     | Hexadecimal:
                                                        abcdH
```

a: Input module type 0: Discrete input module
1: Register input module

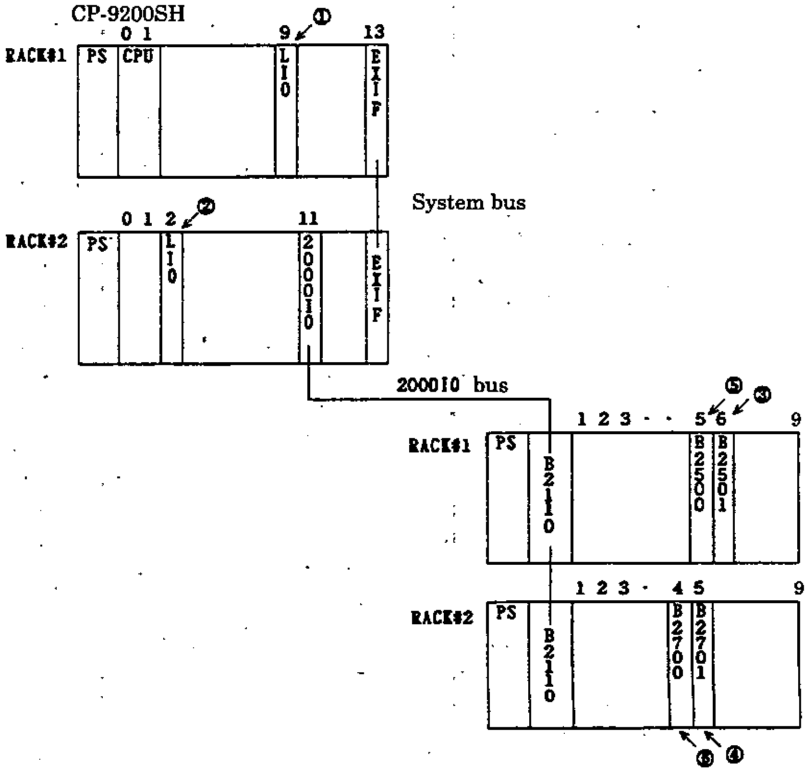
b: Rack number ($1 \leqq b \leqq 4$)

c: Slot number ($1 \leqq c \leqq 9$)

d: Data offset ($0 \leqq d \leqq 7$)

Continuous execution type direct input instruction (INS)

Designation of RSSEL and MDSEL in a system configuration shown below is explained in ex ① to ex ⑥.



ex ① LIO-01 (RACK1/SLOT9) First word is input
RSSEL=0109H MDSEL=0

ex ② LIO-01 (RACK2/SLOT2) Second word is output
RSSEL=0202H MDSEL=1

ex ③ B2501 (Discrete input) (RACK1/SLOT6) connected to 2000IOIF (RACK2/SLOT11) First word is input
RSSEL=020BH MDSEL=0160H

ex ④ B2701 (Register input) (RACK2/SLOT5) connected to 2000IOIF (RACK2/SLOT11) Fourth word is input
RSSEL=020BH MDSEL=1254H

ex ⑤ B2500 (Discrete output) (RACK1/SLOT5) connected to 2000IOIF (RACK2/SLOT11) First word is input
RSSEL=020BH MDSEL=0150H

ex ⑥ B2700 (Register output) (RACK2/SLOT4) connected to 2000IOIF (RACK2/SLOT11) Seventh word is input
RSSEL=020BH MDSEL=1247H

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Data input from LIO mounted at rack 2, slot 4.

```
├─ H0204                                  ⇒MW00100
├─ 0                                      ⇒MW00101
├─ 1                                      ⇒MW00103

  INS  MA00100
```

* Input data stored in MW00104.

4.3.2 Continuous Execution Type Direct Output Instruction (OUTS)

[Format] [Parameter/head address of the data table]

OUTS [Register address (except for #/C)
Register address (except for #/C) with subscript]

[Description] The OUTS instruction conforming to previously set parameter table contents, continuously performs direct output to a single module. The only module that can apply direct output is the LIO-01/2000IO. If no error at all occurs, B register is OFF. If an error occurred in even a single word, B register turns ON. During operation, interruption by the system is prohibited.

Table 4.8 OUTS Instruction Parameter/Data Table

ADR	Type	Symbol	Name	Specifications	Input or Output
0	W	RSSEL	Module designation 1	Designation of module for performing output	IN
1	W	MDSEL	Module designation 2	(The details are described (1) and (2) below.)	IN
2	W	STS	Status	Status for each word output with bit response	OUT
3	W	N	Number of words	Designation of number of continuous output words	IN
4	W	OD1	Output data 1	Setting output data	IN
⋮	⋮	⋮	⋮		⋮
N+3	W	ODN	Output data N		IN

* Method of setting RSSEL and MDSEL is the same as for INS.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Two words output to LIO-01 mounted at rack 3, slot 10.

```
|- H030A                          ⇒MW00200
|- 0                              ⇒MW00201
|- 2                              ⇒MW00203

   Output data 1
|- xxxxx                          ⇒MW00204

   Output data 2
|- yyyyy                          ⇒MW00205

   OUTS   MA00200
```

4.4 Sequence Circuit Instructions

The circuit elements shown in Table 4.9 are used in combination to prepare sequence circuits.

Table 4.9 Sequence Circuit Elements

No.	Sequence Circuit Element	Symbol	Remarks
1	N.O. contact instruction	⊣⊢	Connection indication elements (1) Branching ⊤ (2) Parallel connection point ⊤ (3) Parallel connection ⤴
2	N.C. contact instruction	⊣/⊢	
3	Coil	—○┤	
4	Set coil	⊣S⊢	
5	Reset coil	⊣R⊢	
6	Rising pulse	⊣↑⊢	
7	Falling pulse	⊣↓⊢	
8	On-delay timer (10ms unit)	⊣ ⊢	
9	Off-delay timer (10ms unit)	⊣ ⊢	
10	On-delay timer (1s unit)	⊣ ⊢	
11	Off-delay timer (1s unit)	⊣ ⊢	

4.4.1 N.O. Contact Instruction (⊣⊢)

[Format]

[Any bit type register
Any bit type register
with subscript]

——⊣⊢——

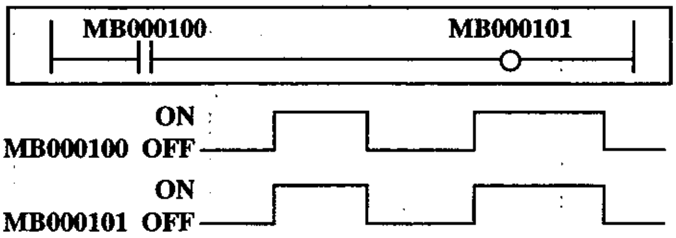
[Description] The N.O. contact instruction sets the status of the B register to ON if the value of th referenced register is 1 (ON) and to OFF if the value of the referenced register is (OFF).

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

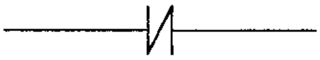
[Example(s)] When MB000100 becomes ON, MB000101 becomes ON.



.4.2 N.C. Contact Instruction (-|/|-)

[Format]

Any bit type register
Any bit type register
with subscript



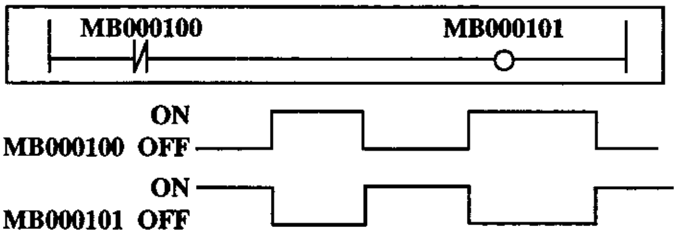
[Description] The N.C. contact instruction sets the status of the B register to OFF if the value of the referenced register is 1 (ON) and to ON if the value of the referenced register is 0 (OFF).

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○ : stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

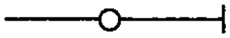
[Example(s)] When MB000100 becomes ON, MB000101 becomes OFF.



4.4.3 Coil Instruction (—o┤)

[Format]

Any bit type register
(except for # and C registers)
Any bit type register with subscript (except for # and C registers)



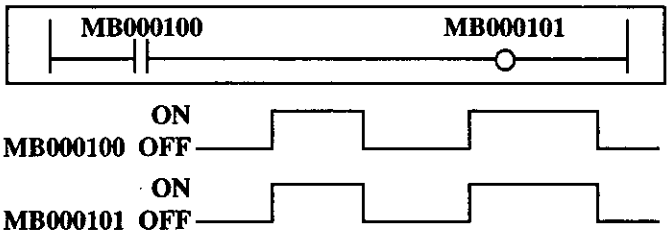
[Description] The coil instruction sets the status of the referenced register to 1 (ON) if the status of the immediately preceding B register is ON and to 0 (OFF) if the status of the immediately preceding B register is OFF.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] When MB000100 becomes ON, MB000101 becomes ON.



4.4.4 Set Coil / Reset Coil Iinstruction (─[S]─┤ / ─[R]─┤)

[Format]

Set coil [Any bit type register (except for # and C registers) Any bit type register with subscript (except for # and C registers)]

───[S]───┤

Reset coil [Any bit type register (except for # and C registers) Any bit type register with subscript (except for # and C registers)]

───[R]───┤

[Description] The set coil instruction turns the output ON when execution conditions are satisfie and maintains that ON status. Conversely, the reset coil instruction turns the outp OFF when execution conditions are satisfied, and maintains that OFF status.

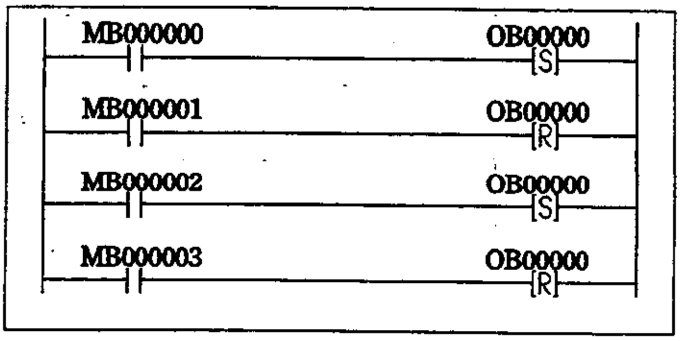
[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

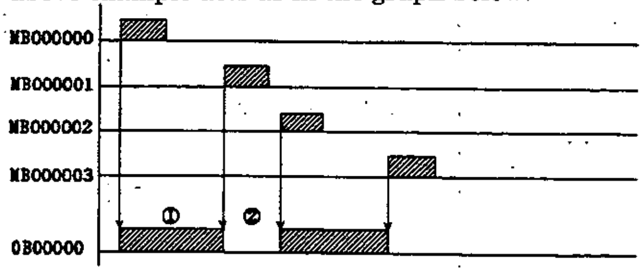
○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]

<Example 1> Case where the same output destination is designated multiple times.

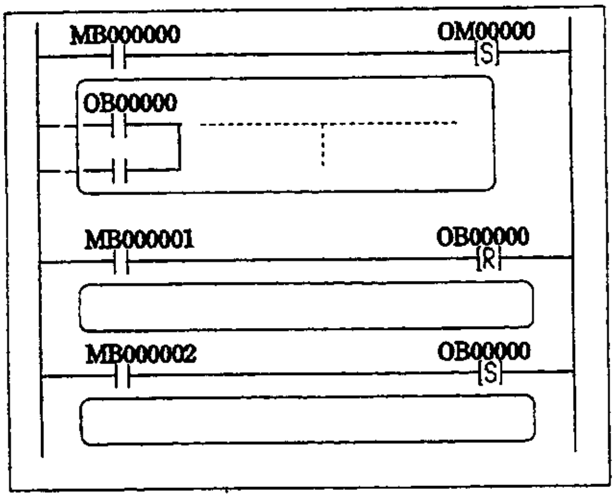


The above example acts as in the graph below.



(1) When OB00000 is OFF, with the "set coil" instruction, OB00000 turns ON.
(2) When OB00000 is ON, with the "reset coil" instruction, OB00000turns

<Example 2> When all execution conditions are ON.



This part of the program is processed assuming OB00000 is ON.

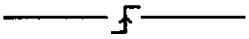
OB00000 is processed as OFF.

OB00000 is processed as ON.

During operation processing, the contents of the output are rewritten with each step.
In the above case, OB00000 is ultimately ON.

4.4.5 Rising Pulse Instruction (–⌐–)

[Format] [Any bit type register (except for # and C registers)
Any bit type register with subscript (except for # and C registers)]



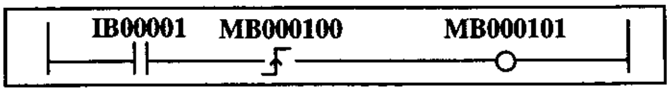
[Description] With the rising pulse instruction, when the status of the immediately preceding B register changes from OFF to ON, the status of the B register turns ON and stays ON during one scan. The designated register is used for storage of the previous value of the B register.

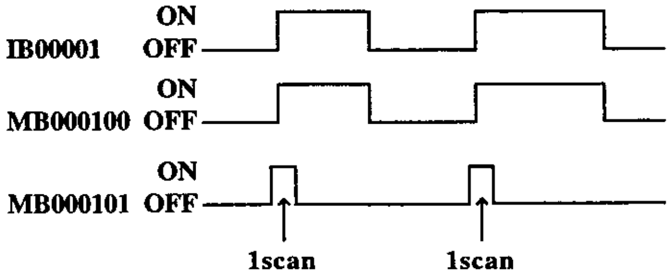
[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] When IB00001 turns ON from OFF, MB000101 turns ON and stays ON during 1 scan. MB000100 is used to store the previous value of IB00001.





Rising Pulse Instruction (—↑—)
Falling Pulse Instruction (—↓—)

Table 4.10 Register Status with Rising Pulse Instruction

Input		Result	
	MB000100	MB000100	
IB00001	(Previous value of IB00001)	(IB00001 stored)	MB000101
OFF	OFF	OFF	OFF
OFF	ON	OFF	OFF
ON	OFF	ON	ON
ON	ON	ON	OFF

NOTE
In the above example, the instruction is used not for rise detection of MB000100 b is used for rise detection of IB00001. MB000100 is used only for storing the previou value of IB00001.
Please be careful not to make a mistake.

4.4.6 Falling Pulse Instruction (—↓—)

[Format] [Any bit type register (except for # and C registers)
Any bit type register with subscript (except for # and C registers)]

——↓——

[Description] With the falling pulse instruction, when the status of the immediately preceding B register changes from ON to OFF, the status of the B register turns ON and stays ON during 1 scan. The designated register is used for storage of the previous value of the B register.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] When IB00001 turns OFF, MB000101 turns ON and stays ON during 1 scan. MB000100 is used to store the previous value of IB00001.

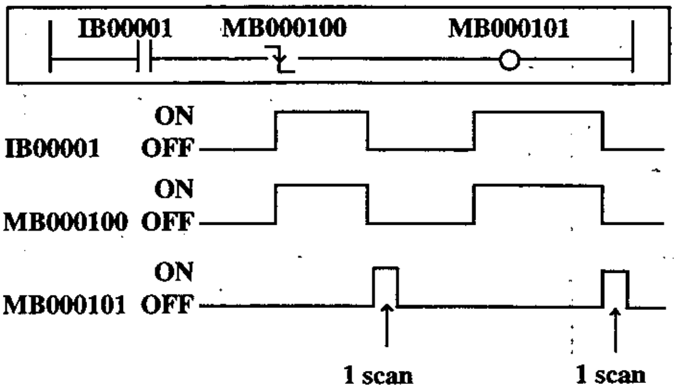


Table 4.11 Register Status with Falling Pulse Instruction

Input		Result	
	MB000100	MB000100	
IB00001	(Previous value of IB00001)	(IB00001 stored)	MB000101
OFF	OFF	OFF	OFF
OFF	ON	OFF	ON
ON	OFF	ON	OFF
ON	ON	ON	OFF

NOTE

In the above example, the instruction is used not for fall detection of MB000100 but is used for fall detection of IB00001. MB000100 is used only for storing the previous value of IB00001.

Please be careful not to make a mistake.

4.4.7 On-delay Timer Instruction: unit of measurement=0.01 seconds (┤ᵀ ├)

[Format] ┤ᵀSet value Count value├

Set value : constant, any integer type register, or any integer type register with subscript (0 to 655.35sec : in 0.01sec unit)

Count value: any integer type register (except for # and C registers), any integer type register with subscript (except for # and C registers)

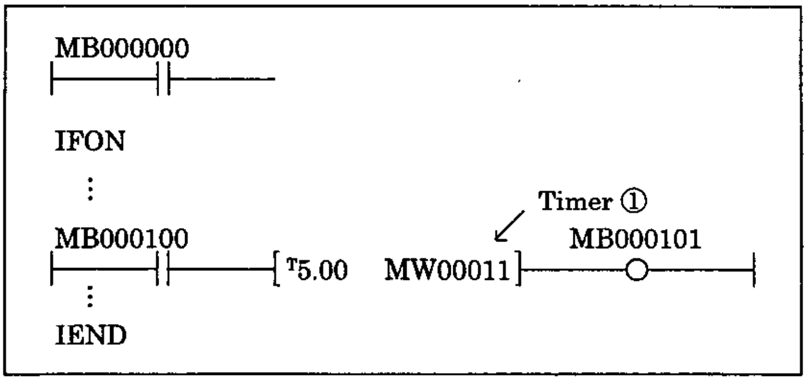
[Description] With the on-delay timer instruction, the time is counted while the status of the immediately preceding B register is ON. The status of the B register becomes ON when "Count value = Set value".

The timer operation is stopped when the status of the immediately preceding B register becomes OFF in the middle of counting. When the B register turns ON again, the counting is started from the beginning (0.00s).

A value equal to the actual counted time × 100 is stored in the count register.

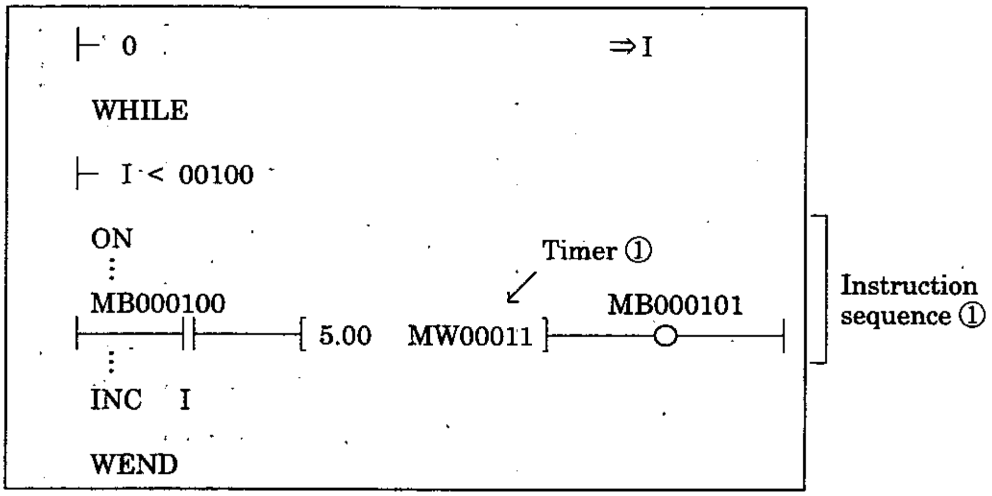
The on-delay timer instruction (┤ᵀ ├) counts when the instruction is executed. Thus, exercise caution when using it in IF, WHILE, or FOR statement.

(1) When used in IF structure statement.



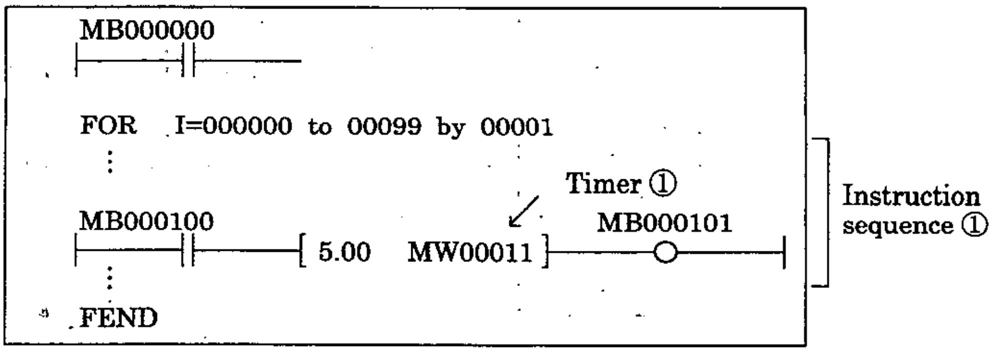
In the above example, when MB000000 is OFF, the instruction of timer ① is not executed, accordingly time is not counted. The time operation remains stopped.

(2) When used in WHILE structure statement



In the above example, since instruction sequence ① is executed 100 times (0≦ ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 1(× scan time set value, so time is counted faster than real time.

(3) When used in FOR structure statement



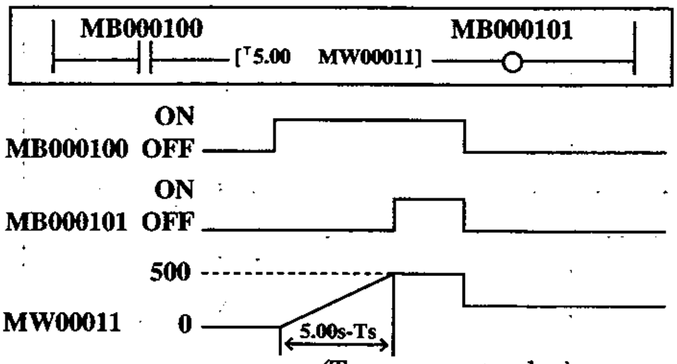
In the above example, since instruction sequence ① is executed 100 times (0≦ ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 1(× scan time set value, so time is counted faster than real time.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]



(Ts = scan set value)

NOTE
MW00011 works as timer count register. Thus, it is essential that there is no overla
Set an unused register.

.4.8 Off-delay Timer Instruction: unit of measurement=0.01 seconds (⊣ ⊢)

[Format] ⊣ Set value Count value^T⊢

Set value : constant, any integer type register, or any integer type register with subscript (0 to 655.35sec : in 0.01sec unit)

Count value: any integer type register (except for # and C registers), any integer type register with subscript (except for # and C registers)

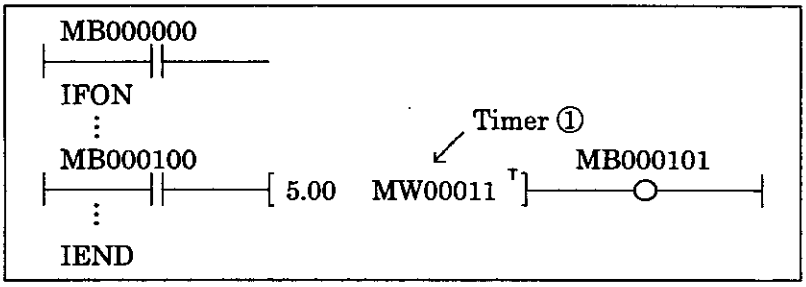
[Description] With the off-delay timer instruction, the time is counted while the status of the immediately preceding B register is OFF. The status of the B register becomes OFF when "Count value = Set value".

The timer operation is stopped when the status of the immediately preceding B register becomes ON in the middle of counting. When the B register turns OFF again, the counting is started from the beginning (0.00s).

A value equal to the actual counted time×100 is stored in the count register.

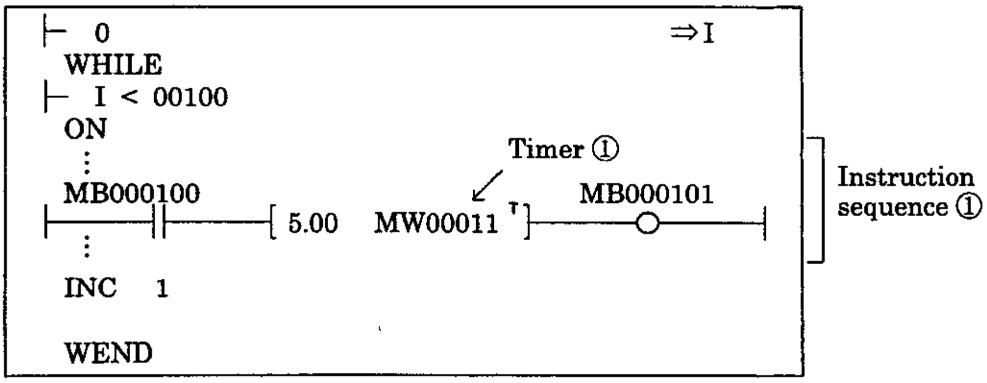
With the off-delay timer instruction, the time is counted when the instruction is executed. 'Therefore, pay attention when using the off-delay instruction in IF, WHILE, and FOR structure statement.

(1) When used in IF structure statement



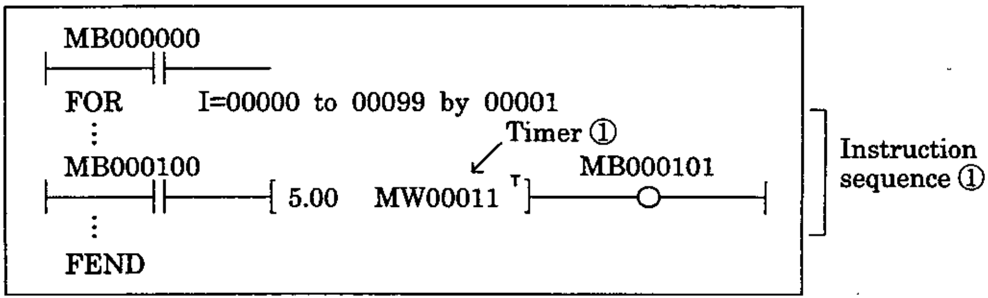
In the above example, when MB000000 is OFF, the instruction of timer ① is not executed, time is not counted. The timer operation remains stopped.

(2) When used in WHILE structune statement.



In the above example, since instruction sequence ① is executed 100 times (0≦I ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

(3) When used in FOR structure statement



In the above example, since instruction sequence ① is executed 100 times (0≦I ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

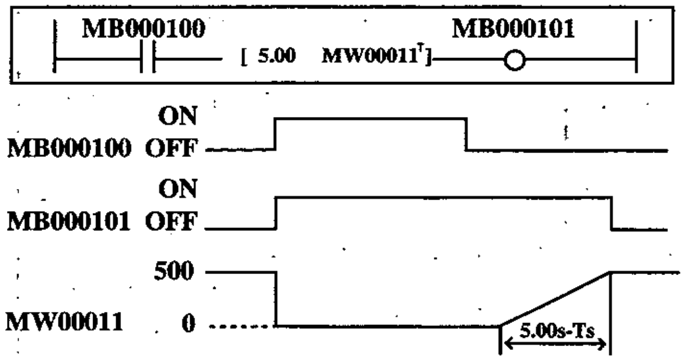
Off-delay Timer Instruction: unit of measurement=0.01 seconds (⊣ ⊢)

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]



(Ts = scan set value)

NOTE

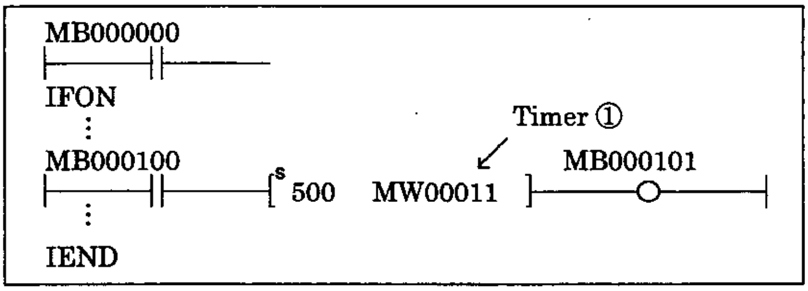
In the above example, MW00011 functions as the count register of the timer. B sure to set an unused register for the count register so that an overlap will n occur.

.4.9 On-delay Timer Instruction: unit of measurement=1 second (-[s]-)

[Format] -[sSet value Count value]-
Set value : constant, any integer type register, or any integer type register with subscript (0 to 65535sec : in 1sec unit)
Count value: any integer type register (except for # and C registers), any integer type register with subscript (except for # and C registers)

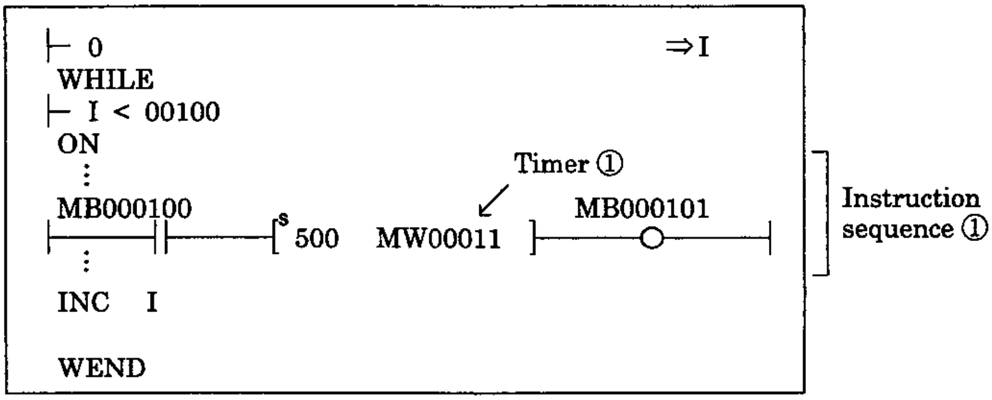
[Description] With the on-delay timer instruction, the time is counted while the status of the immediately preceding B register is ON. The status of the B register becomes ON when "Count value = Set value".
The timer operation is stopped when the status of the immediately preceding B register becomes OFF in the middle of counting. When the B register turns ON again, the counting is started from the beginning (0s).
A value equal to the actual counted time×1 is stored in the count register.
With the off-delay timer instruction, the time is counted when the instruction is executed. Therefore, pay attention when using the on-day instruction in IF, WHILE, and FOR structure statement.

(1) When used in IF structure statement



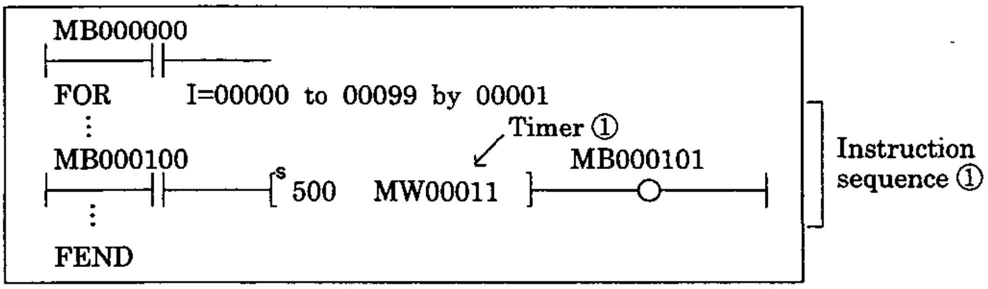
In the above example, when MB000000 is OFF, the instruction of timer ① is not executed, time is not counted. The timer operation remains stopped.

(2) When used in WHILE structure statement.



In the above example, since instruction sequence ① is executed 100 times (0≦I ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

(3) When used in FOR structure statement.



In the above example, since instruction sequence ① is executed 100 times (0≦I ≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

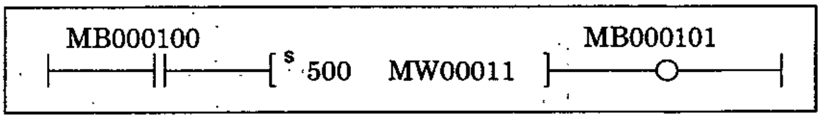
On-delay Timer Instruction: unit of measurement=1 second (⊣^{s} ⊢)

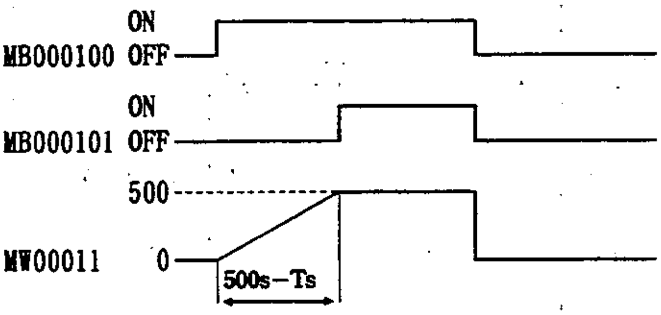
[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]





(Ts = scan set value)

NOTE
In the above example, MW00011 functions as the count register of the timer. Be sure to set an unused register for the count register so that an overlap will not occur.

.4.10 Off-delay Timer Instruction: unit of measurement=1 second (⊣ˢ⊢)

[Format] ⊣ Set value Count value ˢ⊢

Set value : constant, any integer type register, or any integer type register with subscript (0 to 65535sec : in 1sec unit)

Count value: any integer type register (except for # and C registers), any integer type register with subscript (except for # and C registers)

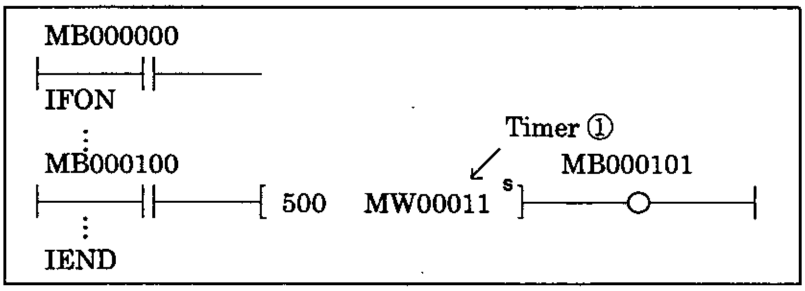
[Description] With the off-delay timer instruction, the time is counted while the status of the immediately preceding B register is OFF. The status of the B register becomes OFF when "Count value = Set value".

The timer operation is stopped when the status of the immediately preceding B register becomes ON in the middle of counting. When the B register turns OFF again, the counting is started from the beginning (0s).

A value equal to the actual counted time×1 is stored in the count register.

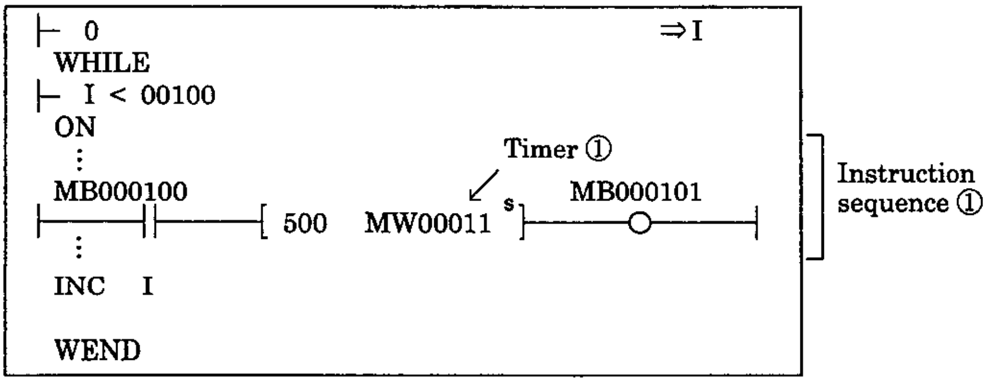
With the on-delay timer instruction, the time is counted when the instruction is executed. Therefore, pay attention when using the on-delay instruction in IF, WHILE, and FOR structure statement.

(1) When used in IF structure statement.



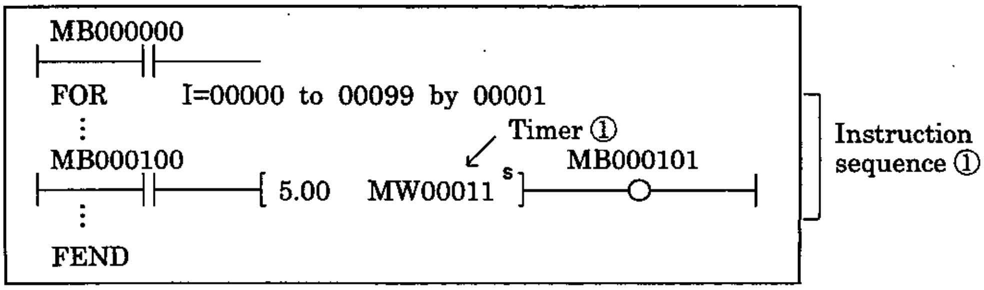
In the above example, when MB000000 is OFF, the instruction of timer ① is not executed, time is not counted. The timer operation remains stopped.

(2) When used in WHILE structure statement.



In the above example, since instruction sequence ① is executed 100 times (0≦I≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

(3) When used in FOR structure statement



In the above example, since instruction sequence ① is executed 100 times (0≦I≦99), the timer ① is also executed 100 times. Thus, the time is counted for 100 × scan time set value, so time is counted faster than real time.

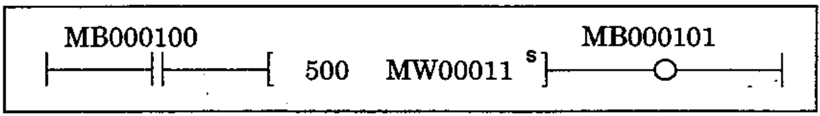
Off-delay Timer Instruction: unit of measurement=1 second (┤ˢ├)

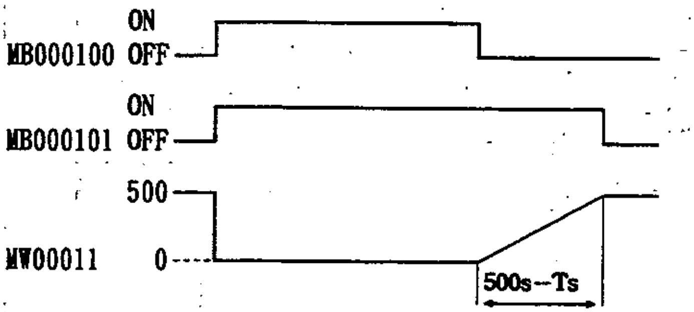
[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]





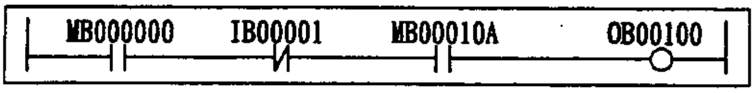
(Ts = scan set value)

NOTE
In the above example, MW00011 functions as the count register of the timer. B sure to set an unused register for the count register so that an overlap will n occur.

Examples of Relay Circuit Combinations

Example of a Series Circuit

In the example below, relays are connected in series and their logical product is output to a coil.

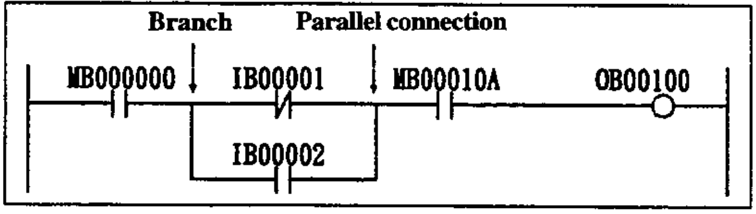


Examples of Branched and Parallel Circuits

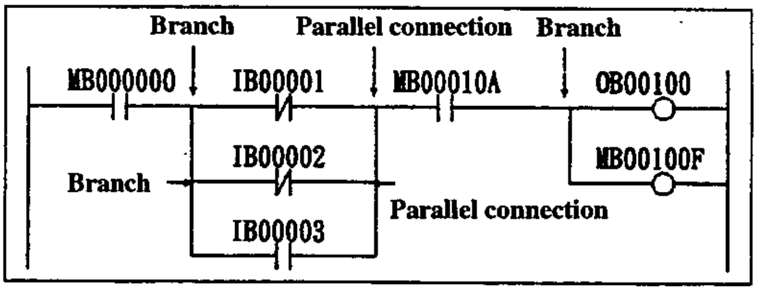
The branch indication element is used to branch the contents of the B register to several parts. The parallel connection indication element is used to determine the logical sum (OR) of a plurality of relays.

In the examples below, relays are connected in series and in parallel and the result is output to a coil or to coils.

(Example 1) Simple example of branching and parallel connection



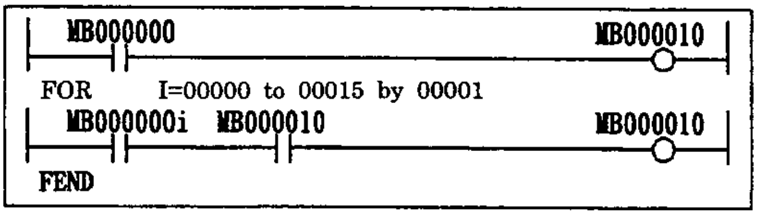
(Example 2) Example in which several branches and parallel connections are used



Example of a Sequence Circuit with Subscript

A relay number may be used with a subscript.

In the example below, the logical product (AND) of relays MB000000 to MB00000F is determined and set in MB000010.



4.5 Logical Operation Instructions

The AND (∧), OR (∨), and XOR (⊕) instructions are available as logical operation instructions.

4.5.1 AND Instruction

[Format] ∧ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant]

[Description] The AND instruction outputs the logical product (AND) of the immediately preceding A register and the designated register to the A register.

1-bit Truth Table for the Logical Product (AND : A ∧ B = C)

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The logical product of MW00100 and a constant is stored in MW00101.

```
├MW00100  ∧  H00FF           ⟹ MW00101
 (H1234)     (H00FF)            (H0034)
```

4.5.2 OR Instruction

[Format] ∨ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant]

[Description] The OR instruction outputs the logical sum (OR) of the immediately preceding A register and the designated register to the A register.
1-bit Truth Table for the Logical Sum (OR : A ∨ B = C)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The logical sum of MW00100 and a constant is stored in MW00101.

```
⊢MW00100 ∨ H00FF          ⟹ MW00101
 (H1234)    (H00FF)          (H12FF)
```

4.5.3 XOR Instruction

[Format] ⊕ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant]

[Description] The XOR instruction outputs the exclusive logical sum (XOR) of the immediately preceding A register and the designated register to the A register.
1-bit Truth Table for the Exclusive Logical Sum (XOR : A ⊕ B = C)

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The exclusive logical sum of MW00100 and a constant is stored in MW00101.

```
⊢MW00100 ⊕ H00FF          ⟹ MW00101
 (H5555)    (H00FF)          (H55AA)
```

4.6 Numerical Operation Instructions

Data types include the integer type, the double-length integer type, and the real number type. Refer the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details.

4.6.1 Integer Type Entry Instruction

[Format] ⊢ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant]

[Description] The integer type entry instruction enters data into the A register and starts an integ type operation. There on after, real number type data cannot be used until a real numb type entry instruction appears.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The contents of MW00100 are entered in the A register.

```
⊢MW00100
```

The contents of ML00100 are entered in the A register.

```
⊢ML00100
```

```
⊢MW00100                         ⟹ MW00200
  (01234)                            (01234)
⊢MW00101                         ⟹ MW00201
  (00001)                            (00001)
⊢ML00100                         ⟹ ML00200
 (66770)                            (66770)
```

ML00100=66770 Lower 16 bits : MW00100 = 01234 = H04D2
Upper 16 bits : MW00101 = 00001 = H0001

.6.2 Real Number Type Entry Instruction (⊩)

[Format] ⊩ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Any real number type register
Any real number type register with subscript
Subscript register
Constant]

[Description] The real number type entry instruction enters data into the F register and starts a real number type operation. The series of operations beginning with a real number type entry instruction can be programmed using integer, double-length integer, and real number type registers. When an integer or double-length integer type register is designated for a real number type entry instruction, the data is automatically converted to a real number type data upon execution.

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The content of DF00200 are entered in the F register.

```
⊩ DF00200
```

The integer type data in DW00100 are converted to real number type data and then stored in the F register.

```
⊩ DW00100
```

The double-length integer type data in DL00100 are converted to real number type data and then stored in the F register.

```
⊩ DL00100
```

```
⊩ DW00000              ⟹ DF00010
   (00001)                (1.0E+00)
⊩ DL00001              ⟹ DF00012
   (1234567)              (1.234567E+06)
⊩ DF00004              ⟹ DF00014
   (-2.5E+00)             (-2.5E+00)
```

NOTE
The following form of usage is not allowed.

```
⊢ 12345                    ⟹ DF00200
```

4.6.3 Storage Instruction

[Format] ⟹ [Any integer type register (except for # and C-registers)
Any integer type register with subscript (except for # and C registers)
Any double-length integer type register (except for # and C registers)
Any double-length integer type register with subscript (except for # and C registers)
Any real number type register (except for # and C registers)
Any real number type register with subscript (except for # and C registers)
Subscript register]

[Description] The storage instruction stores the contents of the F register or the A register in t designated register. Whether the A register or the F register is selected is determin by the type of the immediately preceding entry instruction.

⊢ (Integer entry instruction) ⟹ The contents of the A register are store
⊩ (Real number entry instruction) ⟹ The contents of the F register are store

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored ✕ : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The contents of the A register are stored in MW00100.

```
⊢ 12345                          ⟹ MW00100
```

The contents of the A register are stored in ML00100.

```
⊢ 1234567                        ⟹ ML00100
```

The contents of the F register are stored in DF00100 as they are in the real number for

```
⊩ 1.23456                        ⟹ DF00100
                                    (1.23456)
```

The contents of the F register are converted into integer form and then stored in DW0010

```
⊩ 1.234567                       ⟹ DW00100
                                    (00001)
```

The contents of the F register are converted into double-length integer form and stored in DL001

```
⊩ 123456.7                       ⟹ DL00100
                                    (123457)
```

NOTE

(1) The following form of usage is not allowed.

```
⊢ 12345                          ⟹ DF00200
```

(2) When a double-length integer type data is stored in an integer type register, tl lower 16 bits are stored as they are. Be careful since an operation error will n occur even if the data to be stored exceeds the integer range (-32768 to 32767)

```
⊢ ML00100                        ⟹ MW00200
  (65535)                           (−00001)
```

.6.4 Addition Instruction (+)

[Format] +
- Any integer type register
- Any integer type register with subscript
- Any double-length integer type register
- Any double-length integer type register with subscript
- Any real number type register
- Any real number type register with subscript
- Subscript register
- Constant

[Description] The addition instruction performs addition of integer type, double-length integer type, and real number type values. An overflow operation error will occur if the result of addition of integer type values is greater than 32767. An overflow operation error will occur if the result of addition of double-length integer type values is greater than 2147483647.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○ : stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .

*2: Will not be stored if the operation starts with a ‖⊢ . Will be stored if the operation does not start with a ‖⊢ .

[Example(s)] Addition of integer type values

```
⊢MW00100+12345           ⟹ MW00101
  (03000)                   (15345)
⊢ML00102+ML00104         ⟹ ML00106
  (100000)  (200000)        (300000)
```

Addition of real number type values

```
‖⊢DF00200+1.23456        ⟹ DF00202
   (10.0)                   (11.23456)
‖⊢DF00204+DW00206        ⟹ DF00208
   (0.15)     (00006)       (6.15)
‖⊢DF00210+DL00212        ⟹ DF00214
   (3.51)     (100000)      (100003.51)
```

NOTE

In the case of double-length integer type values, an operation using addition and subtraction instructions (+, -, ++, --) will be a 32-bit operation. However, when an addition or subtraction instruction is used in a remainder correction operation (where a multiplication instruction (×) is the immediately preceding instruction and a division instruction (÷) is the immediately subsequent instruction), the operation will be a 64-bit operation.

Remainder correction operation $(y) = \frac{a \times b + c}{d}$

```
      a          b          c          d
⊢ ML00400×ML00402 + ML00404÷ML00406            y
                                          ⟹ ML00408
                                               c
  MOD                                     ⟹ ML00404
```

4.6.5 Subtraction Instruction (-)

[Format] - [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Any real number type register
Any real number type register with subscript
Subscript register
Constant]

[Description] The subtraction instruction performs subtraction of integer type, double-length integ type, and real number type values. An underflow operation error will occur if t subtraction result of integer type values is less than -32768. An underflow operati error will occur if the subtraction result of double-length integer type values is le than -2147483648.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored ✕ : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2: Will not be stored if the operation starts with a ‖⊢ . Will be stored if the operation does not start with a ‖⊢ .

[Example(s)] Subtraction of integer type values

```
⊢ MW00100 − 12345              ⟹ MW00101
  (03000)                         (−09345)
⊢ ML00102 − ML00104            ⟹ ML00106
  (100000)   (200000)             (−100000)
```

Subtraction of real number type values

```
‖⊢ DF00200 − 1.23456            ⟹ DF00202
   (10.0)                          (8.76544)
‖⊢ DF00204 − DW00206            ⟹ DF00208
   (0.15)      (00006)             (−5.85)
‖⊢ DF00210 − DL00212            ⟹ DF00214
   (3.51)      (100000)            (−99996.49)
```

NOTE

In the case of double-length integer type values, an operation using addition an subtraction instructions (+, -, ++, --) will be a 32-bit operation. However, when a addition or subtraction instruction is used in a remainder correction operation (whe a multiplication instruction (×) is the immediately preceding instruction and division instruction (÷) is the immediately subsequent instruction), the operatio will be a 64-bit operation.

Remainder correction operation $(y) = \frac{a \times b + c}{d}$

```
     a        b         c        d
⊢  ML00400 × ML00402 + ML00404 ÷ ML00406              y
                                              ⟹ ML00408

                                                      c
   MOD                                        ⟹ ML00404
```

.6.6 Extended Addition Instruction (++)

[Format] ++ Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant

* Cannot be used in a real number type operation begins with a real number type entry instruction (⊩).

[Description] The extended addition instruction performs addition of integer type values. An operation error will not occur even if the operation results in an overflow. Otherwise, the extended addition instruction is identical to the addition instruction in function.

Integer type
- Decimal numbers : 0 → 1···32767 → -32768····-1 → 0
- Hexadecimal numbers : 0000 → 0001···7FFF → 8000···FFFF → 0000

Double-length integer type
- Decimal numbers : 0 → 1···2147483647 → -2147483648····-1 → 0
- Hexadecimal numbers : 00000000 → 00000001···7FFFFFFF → 80000000···FFFFFFFF → 00000000

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] This instruction is used in cases where it is desirable that operation errors do not occur in the addition of integer type values.

```
⊢ MW00100+ +00001              ⟹ MW00101
  (32767)                         (−32768)
```

NOTE

In the case of double-length integer type values, an operation using addition and subtraction instructions (+, -, ++, --) will be a 32-bit operation.

However, when an addition or subtraction instruction is used in a remainder correction operation (where a multiplication instruction (×) is the immediately preceding instruction and a division instruction (÷) is the immediately subsequent instruction), the operation will be a 64-bit operation.

Remainder correction operation (y) = $\frac{a \times b + c}{d}$

```
        a          b          c          d                  y
⊢ ML00400×ML00402+ML00404÷ML00406              ⟹  ML00408

                                                  c
MOD                                            ⟹  ML00404
```

4.6.7 Extended Subtraction Instruction (--)

[Format] -- Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Subscript register
Constant

* Cannot be used in a real number type operation begi with a real number type entry instruction (⊩).

[Description] The extended subtraction instruction performs subtraction of integer type values. operation error will not occur even if the operation results in an underflow. Otherwi the extended subtraction instruction is identical to the subtraction instruction in functic

Integer type	Decimal numbers	: 0 → 1···-32767 → 32768···1 → 0
	Hexadecimal numbers	: 0000 → FFFF···8000 → 7FFF···0001 → 00
Double-length integer type	Decimal numbers	: 0 → -1···-2147483648 → -2147483647···1 –
	Hexadecimal numbers	: 00000000 → FFFFFFFF···80000000 → 7FFFFFFF···00000001 → 00000000

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○ : stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] This instruction is used in cases where it is desirable that operation errors do not occ in the subtraction of integer type values.

```
⊢   MW00100- -00001      ⟹  MW00101
    (-32768)                (32767)
```

NOTE

In the case of double-length integer type values, an operation using addition ar subtraction instructions (+, -, ++, --) will be a 32-bit operation.

However, when an addition or subtraction instruction is used in a remaind correction operation (where a multiplication instruction (×) is the immediate preceding instruction and a division instruction (÷) is the immediately subseque instruction), the operation will be a 64-bit operation.

Remainder correction operation $(y) = \frac{a \times b + c}{d}$

```
          a          b          c          d                  y
⊢ MW00400×ML00402+ML00404÷ML00406                  ⟹  ML00408

                                                             c
  MOD                                              ⟹  ML00404
```

4.6.8 Multiplication Instruction (×)

[Format] × [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Any real number type register
Any real number type register with subscript
Subscript register
Constant]

[Description] The multiplication instruction performs multiplication of integer type, double-length integer type, and real number type values. In the case of the multiplication of integer or double-integer type values, × and ÷ are used as a pair. However, if an integer type multiplication result is to be stored in a double-length integer type register, only × is used.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢. Will be stored if the operation does not start with a ⊢.

*2: Will not be stored if the operation starts with a ‖⊢. Will be stored if the operation does not start with a ‖⊢.

[Example(s)] Multiplication of integer type values

```
⊢MW00100×3÷10                 ⟹ MW00101
  (01234)                          (00370)
⊢MW00102×MW00103÷1            ⟹ ML00104
  (00010)   (10000)                (100000)
```

Multiplication of double-length integer type values

```
⊢ML00100×ML00102÷18000        ⟹ ML00104
  (100000)  (009000)               (050000)
⊢ML00106×ML00108÷ML00110      ⟹ ML00112
  (100000)  (100000)  (50000)      (200000)
```

Multiplication of real number type values

```
‖⊢DF00200×DF00100             ⟹ DF00202
   (10.0)    (3.0)                 (30.0)
‖⊢DF00204×DW00206             ⟹ DF00208
   (0.15)    (00002)                (0.3)
‖⊢DF00210×DL00212             ⟹ DF00214
   (0.15)   (100000)               (15000.0)
```

NOTE

With integer type and double-length integer type multiplication, × instruction can be used also independently. However, in this case, make a program so that the result is within 32 bits (-2147483648 to +2147483647). When the result is within 16 bits (-32768 to +32767), it can be stored in integer type register. When the result exceeds 16 bits, store it in double-length integer type register.

```
⊢ MW00100   ×  3              ⟹   MW00101
  (01234)                          (03702)
⊢ MW00102   ×  MW00103        ⟹   ML00104
  (00010)      (10000)             (100000)
⊢ ML00200   ×  ML00202        ⟹   ML00204
  (100000)     (009000)            (900000000)
```

4.6.9 Division Instruction (÷)

[Format] ÷ [Any integer type register
Any integer type register with subscript
Any double-length integer type register
Any double-length integer type register with subscript
Any real number type register
Any real number type register with subscript
Subscript register
Constant]

[Description] The division instruction performs division of integer type, double-length integer ty and real number type values. Although × and ÷ are usually used as a pair, ÷ can a be used alone. Refer to the MOD instruction and the REM instruction concerning t remainder of a division operation. If the value of the designated register is 0, a divisi by-zero error will occur. An operation error will also occur if the result of integer, doub length integer, or real number type division in the F register falls outside the numeri range of the A register.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ├ . Will not be stored if the operation does not start with a ├ .

*2: Will not be stored if the operation starts with a ‖├ . Will be stored if the operation does not start with a ‖├ .

[Example(s)] Division of integer type values

```
├ MW00100×1÷3                  ⟹ MW00101
  (01234)                          (00411)
├ MW00102÷MW00103              ⟹ MW00104
  (01234)    (00003)               (00411)
```

Division of double-length integer type values

```
├ ML00100×ML00102÷ML00110      ⟹ ML00112
  (100000)  (100000)  (50000)     (200000)
├ ML00104÷ML00110              ⟹ ML00114
  (1000000)  (50000)              (000020)
```

Division of real number type values

```
‖├ DF00200÷3.0                 ⟹ DF00202
   (1237.5)                        (412.5)
‖├ DF00200÷DF00204             ⟹ DF00206
   (1237.5)    (3.0)               (412.5)
‖├ DF00200÷DW00208             ⟹ DF00210
   (1237.5)   (00003)              (412.5)
‖├ DF00212÷DL00214             ⟹ DF00216
   (100000.0)  (40000)               (2.5)
```

.6.10 MOD Instruction

[Format] MOD

[Description] The MOD instruction outputs the remainder of an integer type or double-length integer type division to the A register. Execute the MOD instruction immediately after the division instruction or after the storage instruction (⟹). If the MOD instruction is not executed immediately after the division instruction, the remainder of an integer type or double-length integer division will not be guaranteed.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The quotient of an integer type division is stored in MW00101 and the remainder is stored in MW00102.

```
├MW00100  ×  1  ÷  3             ⟹ MW00101
 (00010)                            (00003)
 MOD                             ⟹ MW00102
                                    (00001)
```

The quotient of a double-length integer type division is stored into ML00106 and the remainder is stored in ML00108.

```
├ML00100   ×  ML00102  ÷  ML00104  ⟹ ML00106
 (100000)     (60000)     (34567)     (173575)
 MOD                               ⟹ ML00108
                                      (32975)
```

(Note) : The quotient and remainder are generally determined together. It will thus be convenient to use the instructions in the above manner.

.6.11 REM Instruction

[Format] REM [Any real number type register
Any real number type register with subscript
Constant]

[Description] The REM instruction outputs the remainder of a real number type division to the F register. In this case, the remainder refers to the remainder obtained by repeatedly subtracting the variable value designated by the F register. That is, the output value Y of the REM instruction will be as follows when the F register value is A, the value of the designated variable is X, and the number of repeated subtractions is n:

$$Y = A - (X \times n) \qquad (0 \leqq Y < X)$$

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The remainder of the division of the real number variable MF00200 by the constant value, 1.5, is determined and stored in MF00202.

```
||├MF00200  REM  1.5        ⟹ MF00202
   (4.0)                       (1.0)
```

4.6.12 INC Instruction

[Format] INC [Any integer type register (except for # and C registers)
Any integer type register with subscript (except for # and C registers)
Any double-length integer type register (except for # and C registers)
Any double-length integer type register with subscript (except for # and C registers)
Subscript register]

[Description] The INC instruction adds 1 to the designated integer or double-length integer ty register. In the case of an integer type register, an overflow operation error will n occur even if the addition result exceeds 32767. Likewise, an overflow operation err will not occur in the case of a double-length integer type register.

Integer Type
Decimal number : 0 → 1……32767 → —32768…… —1 → 0
Hexadecimal number : 0000 → 0001……7FFF → 8000……FFFF → 0000

Double-length Integer Type
Decimal number : 0 → 1……2147483647 → —2147483648…… —1 → 0
Hexadecimal number : 00000000 → 00000001…7FFFFFFF → 80000000
…FFFFFFFF → 00000000

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type

```
 ├MW00100++1                    ⟹ MW00100
```

⇕ equivalent

```
   INC   MW00100
```

Double-length integer type

```
 ├ML00100++1                    ⟹ ML00100
```

⇕ equivalent

```
   INC   ML00100
```

NOTE
The following form of usage is not allowed.

```
INC   #W00100   (# register)
INC   DF00200   (real number type register)
```

.6.13 DEC Instruction

[Format] DEC [Any integer type register (except for # and C registers)
Any integer type register with subscript (except for # and C registers)
Any double-length integer type register (except for # and C registers)
Any double-length integer type register with subscript (except for # and C registers)
Subscript register]

[Description] The DEC instruction subtracts 1 from the designated integer or double-length integer type register. In the case of an integer type register, an underflow operation error will not occur even if the subtraction result falls below -32768. Likewise, an underflow operation error will not occur in the case of a double-length integer type register.

Integer Type
Decimal number : 0 → −1······ −32768 → 32767······1 → 0
Hexadecimal number : 0000 → FFFF······8000 → 7FFF······0001 → 0000

Double-length Integer Type
Decimal number : 0 → −1······ −2147483648 → 2147483647······1 → 0
Hexadecimal number : 00000000 → FFFFFFFF······80000000 → 7FFFFFFF
······00000001 → 00000000

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type

```
⊢ MW00100 --1                    ⟹ MW00100
```

⇕ equivalent

```
DEC   MW00100
```

Double-length integer type

```
⊢ ML00100 --1                    ⟹ ML00100
```

⇕ equivalent

```
DEC   ML00100
```

> **NOTE**
> The following form of usage is not allowed.
>
> ```
> DEC  #W00100  (# register)
> DEC  DF00200  (real number type register)
> ```

4.6.14 Time Add Instruction (TMADD)

[Format] [Time to be added] [Time to add]

TMADD [Any integer type register (except for # and C registers)
Any integer type register with subscript (except for # and C registers)] , [Any integer type register
Any integer type register with subscript]

[Description] The TMADD instruction performs addition on two time data (seconds, minutes, hour
The second parameter (time to add) is added to the first parameter (time to be adde
and the result is stored in the first parameter. It is essential that the formats
parameters 1 and 2 should be as shown in Table 4.12.

Table 4.12 Parameter Format

Register offset	Data contents	Data range (BCD)
0	Hours/minutes	Upper byte (hours): 0 to 23, Lower byte (minutes): 0 to 59
1	Seconds	0000 to 0059

When the contents of the first parameter, second parameter, and operation result are
the data ranges listed above, the operation is performed normally. After operation, t
B register turns OFF. Conversely, if a parameter has data that exceeds the above rang
"9999H" is stored for the seconds of the parameter and the operation is stopped. Th
the B register turns ON.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The time data in DW0000-DW0001 is added to the time data in MW00100-MW0010

```
                                          DB000100
    TMADD MW00100, DW00000          ─────────○─────┤
```

8 hrs 40 min 32 sec + 1 hs22 min 16 sec = 10 hrs 2 min 48 sec
(MW00100) (MW00101) (DW00000) (DW00001) (MW00100) (MW00101)

	Before execution	After execution
MW00100	0840H	1002H
MW00101	0032H	0048H
DW00000	0122H	0122H
DW00001	0016H	0016H

.6.15 Time Subtraction Instruction (TMSUB)

[Format]	[Time subtracted from]		[Time subtracted]
TMSUB	Any integer type register (except for # and C registers) Any integer type register with subscript (except for # and C registers)	,	Any integer type register Any integer type register with subscript

[Description] The TMSUB instruction makes subtraction between two time data (hour/min/sec). The second parameter (time subtracted) is subtracted from the first parameter (time subtracted from), and the result is stored in the first parameter.
The formats of the first and second parameters must be as shown in Table 4.13.

Table 4.13 Parameter Format

Register offset	Data contents	Data range (BCD)
0	Hours/minutes	Upper byte (hours): 0 to 23, Lower byte (minutes): 0 to 59
1	Seconds	0000-0059

When the contents of the first parameter, second parameter, and operation result are in the data ranges listed above, the operation is performed normally. After opreation, the B register turns OFF. Conversely, if a parmeter has data that exceeds the above range, "9999H" is stored for the seconds of the parameter and the operation is stopped. Then the B register turns ON.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The time data in DW0000-DW0001 is subtracted from the time data in MW00100-MW00101.

```
                                      DB000100
   TMSUB  MW00100, DW00000        ------O------|
```

$\underset{\text{(MW00100)}}{\underline{\text{8 hrs 40 min}}}\ \underset{\text{(MW00101)}}{\underline{\text{32 sec}}} - \underset{\text{(DW00000)}}{\underline{\text{1 hs 22 min}}}\ \underset{\text{(DW00001)}}{\underline{\text{16 sec}}} = \underset{\text{(MW00100)}}{\underline{\text{7 hrs 18 min}}}\ \underset{\text{(MW00101)}}{\underline{\text{16 sec}}}$

	Before execution	After execution
MW00100	0840H	0718H
MW00101	0032H	0016H
DW00000	0122H	0122H
DW00001	0016H	0016H

4.6.16 Time Spend Instruction (SPEND)

[Format] [Time being subtracted from and result] [Time subtracted]

SPEND [Any integer type register (except for # and C registers) Any integer type register with subscript (except for # and C registers)] , [Any integer type register. Any integer type register with subscript]

[Description] The SPEND instruction performs subtraction between two time data (Yr/Mo/Day/H Min/Sec), and computes the elapsed time.
The second parameter (time subtracted) is subtracted from the first parameter (tim subtracted from), and the result is stored in the first parameter.
The formats of the first and second parameters must be as shown in Tables 4.14 an 4.15.

Table 4.14 First Parameter Format

Register offset	Data contents	Data range (BCD)	I/O
0	Year (BCD)	0000 to 0099	IN/OUT
1	Month/Day (BCD)	Upper byte (month): 1 to 12, Lower byte (day): 1 to 31	IN/OUT
2	Hours/minutes (BCD)	Upper byte (hours): 0 to 23, Lower byte (minutes): 0 to 59	IN/OUT
3	Seconds (BCD)	0000 to 0059	IN/OUT
4	Total number of seconds	This is the number of records which is obtained by converting Year/Month/Day/ Hour/Minute/Second, which is the results of operations, to seconds. (Double-length integer)	OUT
5			

Table 4.15 Second Parameter Format

Register offset	Data contents	Data range (BCD)	I/O
0	Year (BCD)	0000 to 0099	IN
1	Month/Day (BCD)	Upper byte (month): 1 to 12, Lower byte (day): 1 to 31	IN
2	Hours/minutes (BCD)	Upper byte (hours): 0 to 23, Lower byte (minutes): 0 to 59	IN
3	Seconds (BCD)	0000 to 0059	IN

When the contents of the first parameter, second parameter, and operation result are in the data ranges listed above, the operation is performed normally. After operation, the B register turns OFF. Conversely, if a parameter has data that exceeds the above range, "9999H" is stored for the seconds of the parameter and the operation is stopped. Then the B register turns ON.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The time elapsed from the time data in MW00100 to MW00103 to the time data in DW00000 to DW00003 is stored to MW00100 to MW00105.

```
                                        DB000100
     SPEND  MW00100, DW00000    ─────────○──────┤
```

98 yrs 5 mos 11 days 15 hrs 4 min 47 sec — 98 yrs 4 mos 2 days 8 hs 13 min 8 sec
(MW00100) (MW00101) (MW00102) (MW00103) (DW00000) (DW00101) (DW00102) (DW00103)

	Before execution	After execution
MW00100	H0098	H0000
MW00101	H0511	H0039
MW00102	H1504	H0651
MW00103	H0047	H0039
MW00104	—	3394299 (Decimal)
MW00105	—	
DW00000	H0098	H0098
DW00001	H0402	H0402
DW00002	H0813	H0813
DW00003	H0008	H0008

NOTE
In the operation results, the year is counted as 365 days and a leap year is not taken into consideration. Also, the number of months is not counted. It is counted in days.

4. 7 Numerical Conversion Instructions

The 6 types of numerical conversion instructions shown in Table 4.16 are made available as instructio for changing the contents of the A register or the F register. These instructions use the contents of t A register or the F register as the input and leaves the operation result in the A register or F registe

Table 4.16 Numerical Conversion Instructions

Numerical Conversion Instruction	Operation			Numerical Conversion Operation
	Integer	Double-length Integer	Real Number	
Sign inversion (INV)	○	○	○	Inverts the sign of the contents of the A registe or F register.
Complement of 1 (COM)	○	○	×	Determines the complement of 1 of the value in the A register.
Absolute value (ABS)	○	○	○	Determines the absolute value of the value in the A register or F register.
BIN conversion (BIN)	○	○	×	Performs BIN conversion of the contents of the register.
BCD conversion (BCD)	○	○	×	Performs BCD conversion of the contents of the A register.
Parity conversion (PARITY)	○	○	×	Counts the number of bits in the A register that are ON.
ASCII conversion 1 (ASCII)	○	×	×	Converts the designated character string to ASCII codes.
ASCII conversion 2 (BINASC)	○	×	×	Converts the binary data in A register to ASCII codes.
ASCII conversion 3 (ASCBIN)	○	×	×	Converts the ASCII codes to binary data and stores them in A register.

4.7.1 INV Instruction

[Format] INV

[Description] Inverts the sign of the contents of the A register or F register.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .

*2: Will not be stored if the operation starts with a ‖⊢ . Will be stored if the operation does not start with a ‖⊢ .

[Example(s)] Integer type data (A register)

```
⊢MW00100  INV                  ⟹ MW00101
 (00100)                          (−00100)
```

Double-length integer type data (A register)

```
⊢ ML00100   INV                ⟹ ML00102
 (100000)                         (−100000)
```

Real number type data (F register)

```
‖⊢ DF00200   INV               ⟹ DF00202
    (1.0)                          (−1.0)
```

4.7.2 COM Instruction

[Format] COM

[Description] Determines the complement of 1 of the value in the A register.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data (A register)

```
⊢MW00100  COM            ⟹ MW00101
 (H5555)                    (HAAAA)
```

Double-length integer type data (A register)

```
⊢ML00100  COM            ⟹ ML00102
 (H55555555)                (HAAAAAAAA)
```

4.7.3 ABS Instruction

[Format] ABS

[Description] Determines the absolute value of the value in the A register or F register.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1 : Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2 : Will not be stored if the operation starts with a ⊩ . Will be stored if the operation does not start with a ⊩ .

[Example(s)] Integer type data (A register)

```
⊢ MW00100  ABS          ⇒ MW00101
  (-00100)                (00100)
```

Double-length integer type data (A register)

```
⊢ ML00100  ABS          ⇒ ML00102
  (-100000)               (100000)
```

Real number type data (F register)

```
⊩ DF00200  ABS          ⇒ DF00202
  (-1.0)                  (1.0)
```

4.7.4 BIN Instruction

[Format] BIN

[Description] This instruction converts a numeral expressed in BCD in the A register into a bina number (BIN conversion). If the (4-digit) numeral expressed in BCD in the integer ty A register is abcd, the output value Y of the BIN instruction can be determined by t following formula:

Y = (a × 1000) + (b × 100) + (c × 10) + d

Although the above formula will be applied even if the numeral in the A register is n of a BCD expression (e.g. 123FH, etc.), a correct result will not be obtained in su cases.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data (A register)

```
⊢ MW00100  BIN          ⇒ MW00101
  (H1234)                 (D01234)
```

Double-length integer type data (A register)

```
⊢ ML00100  BIN          ⇒ ML00102
  (H12345678)             (D12345678)
```

4.7.5 BCD Instruction

[Format] BCD

[Description] This instruction converts a numeral expressed in binary in the A register into a BC expression (BCD conversion). If the (4-digit) decimal expression of the numeral in th integer type A register is 0abcd, the output value Y of the BCD instruction can b determined by the following formula:

Y = (a × 4096) + (b × 256) + (c × 16) + d

Although the above formula will be applied even if the numeral in the A register cann be expressed in BCD (e.g. a number over 9999, negative numbers, etc.), a correct resu will not be obtained in such cases.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data (A register)

```
⊢ MW00100  BCD          ⇒ MW00101
  (D01234)                (D1234)
```

Double-length integer type data (A register)

```
⊢ ML00100  BCD          ⇒ ML00102
  (D12345678)             (H12345678)
```

.7.6 PARITY Instruction

[Format] PARITY

[Description] This instruction is used to compute the number of binary expression bits that are ON (=1) in the A register.

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)]

Integer type data (A register)

```
⊢ MW00100  PARITY        ⇒ MW00101
  (HF0F0)                   (00008)
```

Double-length integer type data (A register)

```
⊢ ML00100  PARITY        ⇒ MW00102
  (HF0F0F0F0)               (00016)
```

.7.7 ASCII Instruction

[Format]

[Storage register number] [Text]

ASCII [Any integer type register (except for # and C registers)
Any integer type register with subscript (except for # and C registers] " [ASCII characters] "

[Description] The ASCII instruction converts the specified character string in the instruction to ASCII codes, and stores them in the designated storage register.
These are stored in the order: first character, lower byte of the first word, second character, upper byte of the first word. If the length of the character string is odd, the upper byte of the last word in the storage register is a 0. A maximum of 32 characters may be entered.

ASCII VW□□□□□ ⇐ "character string"

	Upper byte	Lower byte	
VW□□□□□	Second character	First character	
VW□□□□□+1	Fourth character	Third character	
VW□□□□□+2	Sixth character	Fifth character	
VW□□□□□+3	Eighth character	Seventh character	V=S, I, O, M, D
		n th character	

If the length of the character string is odd, the upper byte of the last word in the storage register is a 0.

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] (1) The character string "ABCD" is stored in MW00100 to MW00101.

```
ASCII    MW00100      "ABCD"
```

	Upper	Lower	
MW00100	42H ('B')	41H ('A')	MW00100=4241H
MW00101	44H ('D')	43H ('C')	MW00101=4443H

[Format] (2) The character string "ABCDEFG" is stored in MW00100 to MW00103.

```
ASCII    MW00100      "ABCDEFG"
```

	Upper	Lower	
MW00100	42H ('B')	41H ('A')	MW00100=4241H
MW00101	44H ('D')	43H ('C')	MW00101=4443H
MW00102	46H ('F')	45H ('E')	MW00102=4645H
MW00103	00H	47H ('G')	MW00103=0047H

A "0" is entered in the extra byte.

4.7.8 BINASC Instruction

[Format] [Storage register number]

BINASC [Any integer type register
(except for # and C registers)
Any integer type register with subscript
(except for # and C register)]

[Description] The BINASC instruction converts the 16-bit binary data stored in the A register to four digit hexadecimal ASCII code and stores it in the designated storage register (tw words).

```
├ HXYZW (Hexadecimal input data)
              (Storage register)
In the case of BINASC  VW□□□□□
```

	Upper byte	Lower byte	
VW□□□□□	Third digit (Y)	Fourth digit (X)	
VW□□□□□+1	First digit (W)	Second digit (Z)	V=S, I, O, M, D

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The "1234H" binary data stored in the A register is converted to a four digit hexadecim ASCII code and stored in MW00100 to MW00101.

```
├ H1234
  BINASC    MW00100
```

	Upper byte	Lower byte	
MW00100	32H ('2')	31H ('1')	MW00100=3231H
MW00101	34H ('4')	33H ('3')	MW00101=3433H

.7.9 ASCBIN Instruction

[Format] [Storage register number]

ASCBIN [Any integer type register
Any integer type register
with subscript]

[Description] The ASCBIN instruction converts a numerical value expressed in a four digit hexadecimal ASCII code to 16-bit binary data. The converted result is stored in the A register.

In the case of ASCBIN VW□□□□□ (Conversion source register)

Conversion source register

	Upper byte	Lower byte
VW□□□□□	Third digit (Y)	Fourth digit (X)
VW□□□□□+1	First digit (W)	Second digit (Z)

→

A register

Upper	Lower
XY	ZW

V=S, I, O, M, D

[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The four-byte ASCII code stored in MW00100 to MW00101 is converted to two-byte binary data, and the result is stored in MW00200.

```
ASCBIN    MW00100                  ⇒MW00200
```

Data to be converted

	Upper	Lower
MW00100	32H ('2')	31H ('1')
MW00101	34H ('4')	33H ('3')

→ MW00200

A register

Upper	Lower
12H	34H

4.8 Numerical Comparison Instructions

4.8.1 Comparison Instructions

There are 6 types of comparison instructions for comparing numerals and inspecting equivalen relationships.

[Format]

<	Any integer type register
≦	Any integer type register with subscript
=	Any double-length integer type register
≠	Any double-length integer type register with subscript
≧	Any real number type register
>	Any real number type register with subscript
	Subscript register
	Constant

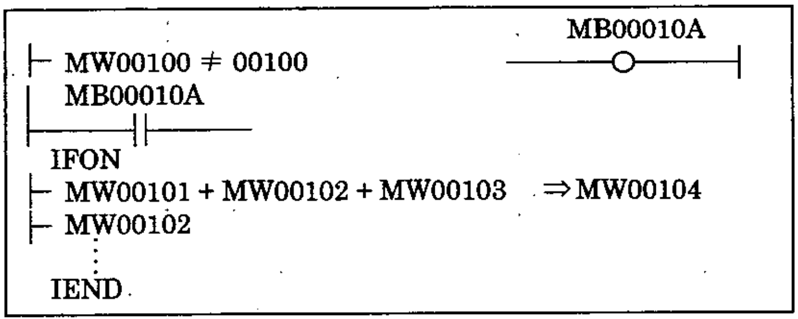
[Description] A comparison instruction stores the result of comparison of the immediately precedi A or F register and the designated register in the B register (ON when true).

[Operation of the Register]

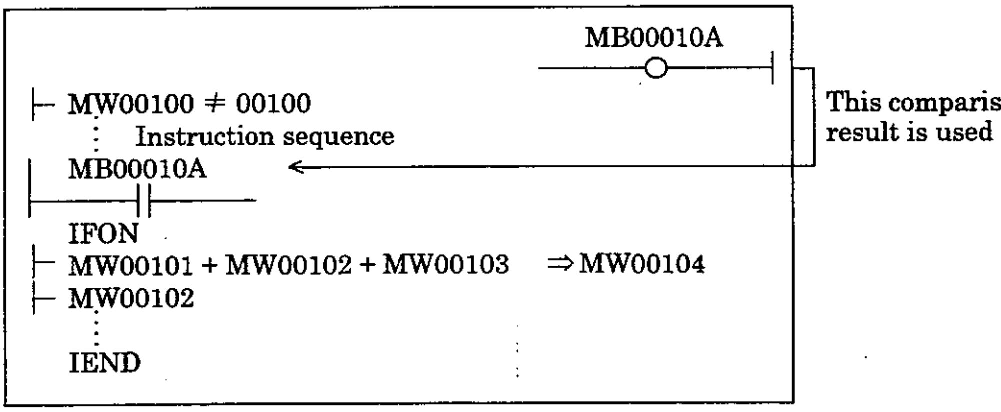
A	F	B	I	J
○	○	×	○	○

○ : stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] (1) If the value of MW00100 is not 100, the instructions from IFON and below a executed.

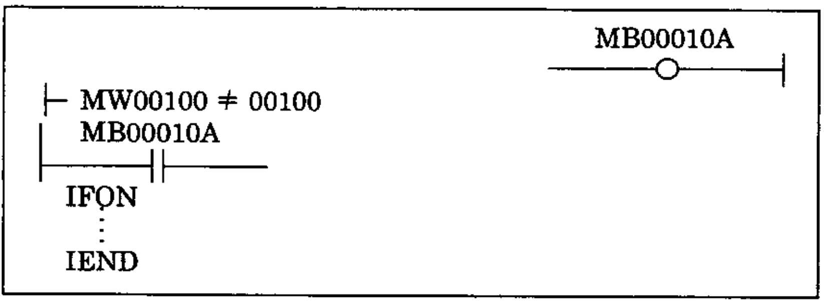


(2) If you want to use the comparison result in a subsequent instruction, it is convenie to accept the comparison result with the coil. Unless the value of MW00100 is 10 MW00010A is set to ON.

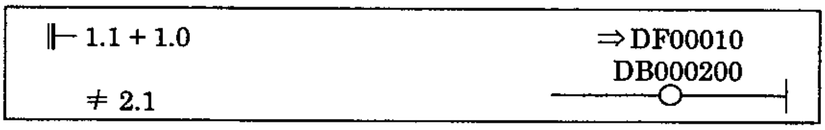


NOTE

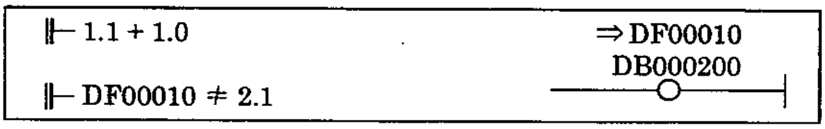
1. Use the NO contact instruction if an IFON (IFOFF) or ON (OFF) instruction is to be used after receiving the comparison result with a coil.



2. When making a comparison of real number type registers, use a ⊩ instruction before the comparison instruction.



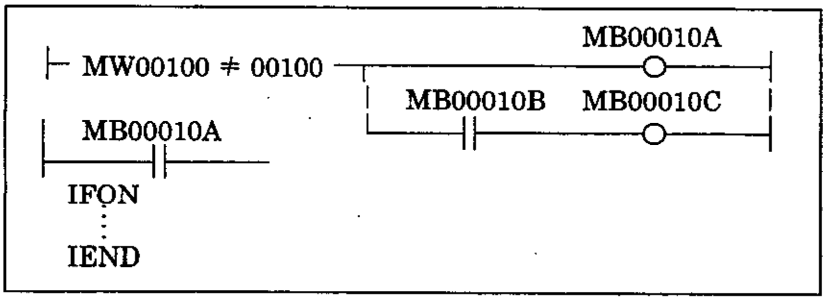
Wrong



Correct

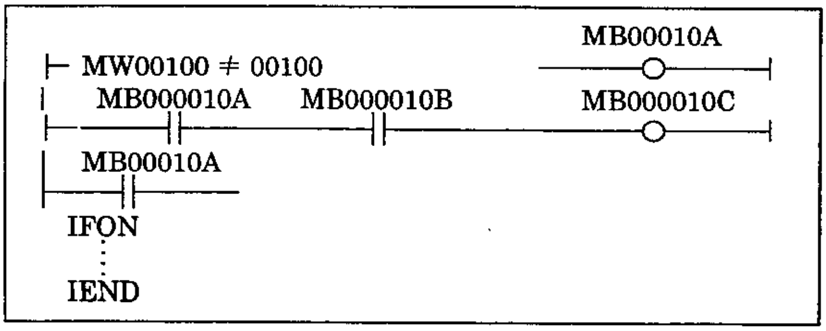
3. In the case of real number type data, since there is a minute precision difference in the data displayed on the CP-717, the execution result of a comparison instruction may not coincide with an apparent result.

4. Do not use instructions other than coil instruction when receiving the comparison result with a coil.



Wrong

↓



Correct

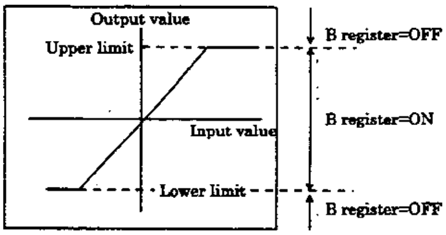
4.8.2 Range Check Instruction (RCHK)

[Format]

[Lower limit]	,	[Upper limit]
Any integer type register		Any integer type register
Any integer type register with subscript		Any integer type register with subscrip
Any double-length integer type register		Any double-length integer type registe
Any double-length integer type register with subscript		Any double-length integer type registe with subscript
Any real number type register		Any real number type register
Any real number type register with subscript		Any real number type register wit subscript
Subscript register		Subscript register
Constant		Constant

[Description] The RCHK instruction examines the contents entered in the A register whether it i within the specified range or not. The result is output to the B register. The content of the A register are kept.

```
( ⊢ input value)
                                                Result
RCHK   [Lower limit], [Upper limit]   ─────────O──┤
```



* If the input value (A register) is greater than the lower limit and less than th upper limit, the result (B register) = ON.

* In the cases other than the above, the result (B register) = OFF.

[Operation of the Register]

A	F	B	I	J
○	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] ■ For integer type operation

```
├ MW00100
                                   DB000000
  RCHK  −1000,   1000   ───────────O──────┤
```

Input (MW00100)	Output (DB000000)
—1000>MW00100	OFF
—1000≦MW00100≦1000	ON
MW00100>1000	OFF

■ For double-length integer type operation

```
├ ML00100
                                   DB000000
  RCHK  −100000,   100000 ─────────O──────┤
```

Input (ML00100)	Output (DB000000)
—100000>ML00100	OFF
—100000≦ML00100≦100000	ON
ML00100>100000	OFF

■ For real number type operation

```
├ DF00100
                                              DB000000
   RCHK   −10.5,    10.5        ──────────────○────────┤
```

Input (DF00100)	Output (DB000000)
—10.5>DF00100	OFF
—10.5≦DF00100≦10.5	ON
DF00100>10.5	OFF

4.9 Data Operation Instructions

4.9.1 ROTL Instruction and ROTR Instruction

[Format]

	[Head Bit Address]		[Number of Rotations]		[Bit Width]
ROTL ROTR	Any bit type register (except for # and C registers) Any bit type register with subscript (except for # and C registers)	N=	Any integer type register Any integer type register with subscript Constant	W=	Any integer type regist Any integer type regist with subscript Constant

[Description] The ROTL (or ROTR) instruction is used to perform rotation, in the left (or right) directi for the number of times designated, on the bit table designated by the head bit addre and the bit width.

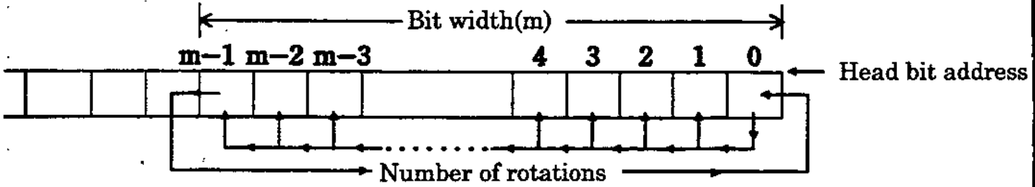


Fig. 4.5 The ROTL Operation

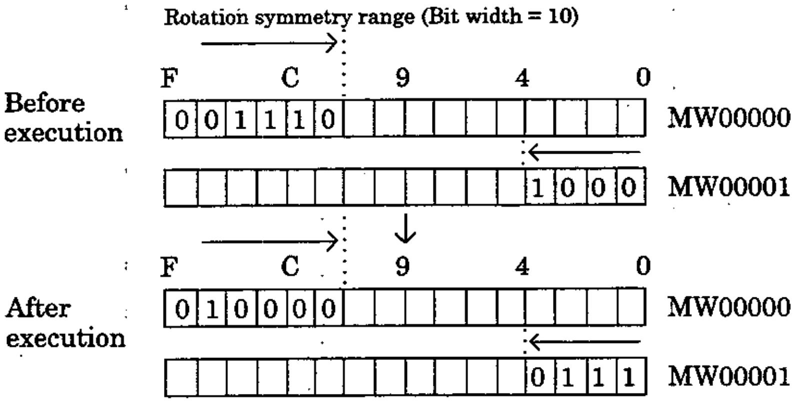
[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

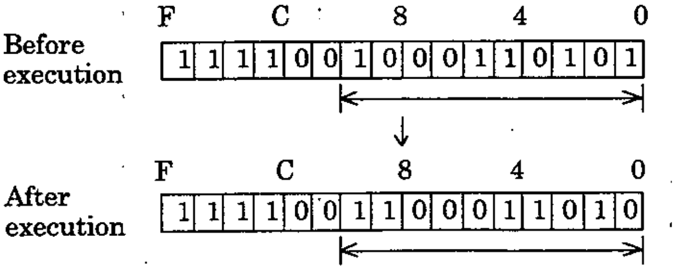
[Example(s)] (1) ROTL The data having MB00000A (bit A of MW00000) as the head address anc bit width of 10 are rotated five times to the left.

```
ROTL MB00000A N=5 W=10
```



(2) ROTR The data having MB000000 (bit 0 of MW00000) as the head address and bit width of 10 are rotated once to the right.

```
ROTR MB000000 N=1 W=10
```



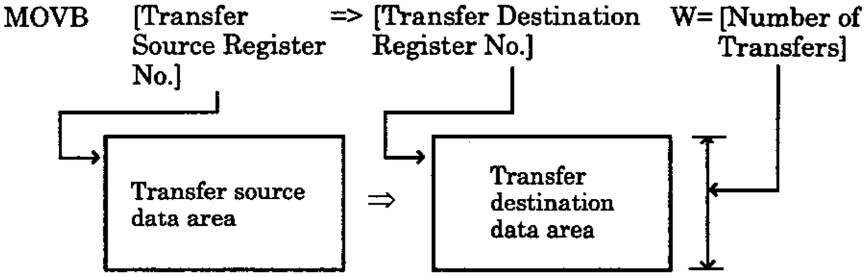
.9.2 MOVB Instruction

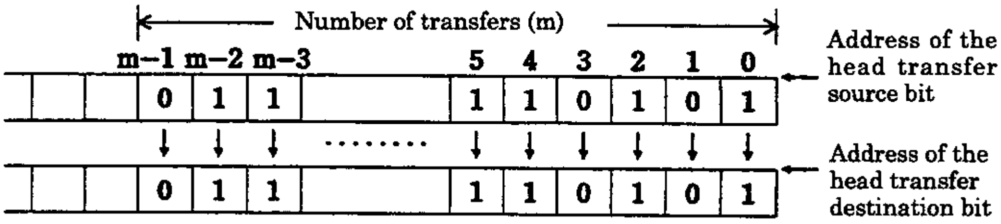
[Format]

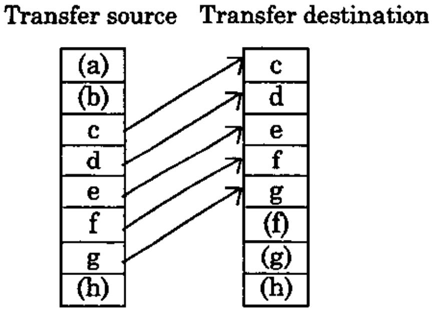
	[Address of Transfer Source Bit]		[Address of Transfer Destination Bit]		[Number of Transfers]
MOVB	Any bit type register Any bit type register with subscript	=>	Any bit type register (except for # and C register) Any bit type register with subscript	W=	Any integer type register Any integer type register with subscript Constant

[Description] The MOVB instruction transfers the designated number of bit data, starting from the head of the transfer source bits, to the transfer destination, which starts from the address of the head transfer destination bit. The transfer is carried out 1 bit at a time in the direction in which the relay number increases.

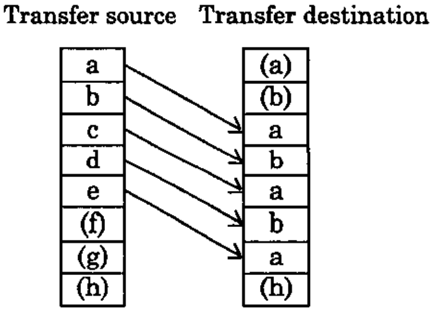
Although the bit table of the transfer source will be stored as long as the transfer source bits and transfer destination bits do not overlap, caution is needed when the bits do overlap.


MOVB
[Transfer Source Register No.]
=> [Transfer Destination Register No.]
W= [Number of Transfers]
Transfer source data area
⇒
Transfer destination data area





When the transfer source and transfer destination overlap (1)



When the transfer source and transfer destination overlap (2)

[Operation of the Register]

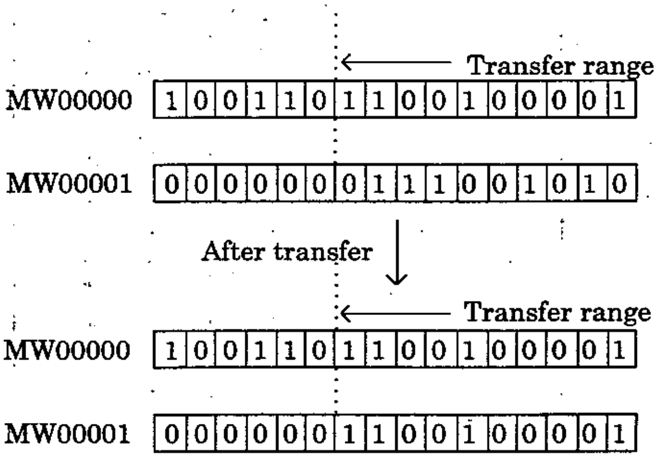
A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

MOVB Instruction

[Example(s)] The 10 bits of data starting from MB000000 (bit 0 of MW00000) are transferred MB000010 (bit 0 of MW00001).

```
MOVB   MB000000 ⇒ MB000010  W=10
```



MW00000	1	0	0	1	1	0	1	1	0	0	1	0	0	0	0	1
MW00001	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	0

After transfer

MW00000	1	0	0	1	1	0	1	1	0	0	1	0	0	0	0	1
MW00001	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1

.9.3 MOVW Instruction

[Format]

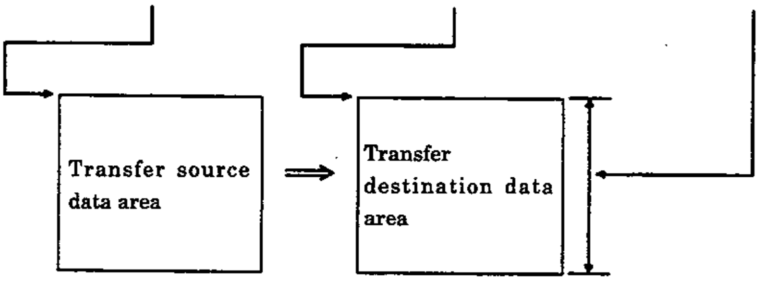
MOVW [Transfer Source Register No.]
- Any integer type register
- Any integer type register with subscript

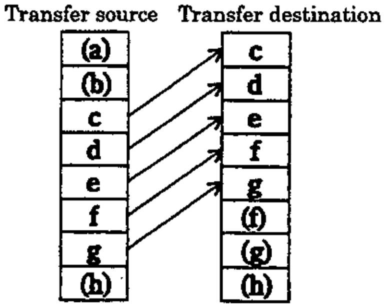
⇒ [Transfer Destination Register No.]
- Any integer type register (except for # and C registers)
- Any integer type register with subscript (except for # and C registers)

W= [Number of Transfers]
- Any integer type register
- Any integer type register with subscript
- Constant

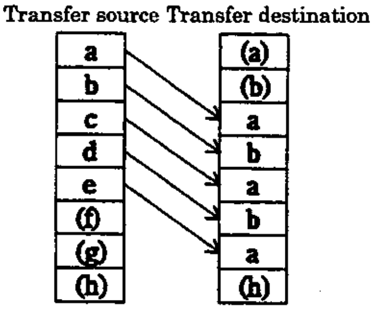
[Description] The MOVW instruction transfers the designated number of words of data, starting from the head of the transfer source registers, to the transfer destination, which starts from the address of the head transfer destination register. The transfer process is carried out 1 word at a time in the direction in which the register number increases. Although the transfer source will be stored as long as the transfer source and the transfer destination do not overlap, caution is needed when these do overlap.

MOVW [Transfer Source Register No.] ⇒[Transfer Destination Register No.] W= [Number of Transfers]





When the transfer source and transfer destination overlap (1)



When the transfer source and transfer destination overlap (2)

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The word data MW00000 to MW00009 are transferred to MW00100 to MW00109.

MOVW MW00000 ⟹ MW00100 W=00010

MW00000	1234H
MW00001	2345H
MW00002	3456H
⋮	⋮
MW00009	9999H

Transfer
⟶

MW00100	1234H
MW00101	2345H
MW00102	3456H
⋮	⋮
MW00199	9999H

4.9.4 XCHG Instruction

[Format]

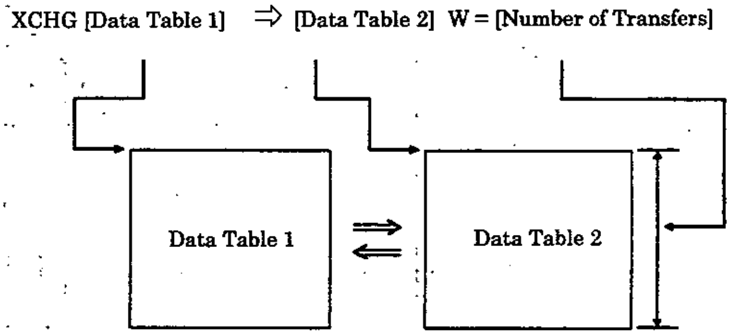
XCHG [Data Table 1] => [Data Table 2] W= [Number of Transfers]

[Data Table 1]
- Any integer type register (except for # and, C registers)
- Any integer type register with subscript (except for # and C registers)

[Data Table 2]
- Any integer type register (except for # and C registers)
- Any integer type register with subscript (except for # and C registers)

[Number of Transfers]
- Any integer type regist
- Any integer type regist with subscript
- Constant

[Description] The XCHG instruction is used to exchange the contents of data table 1 and data table


XCHG [Data Table 1] ⇒ [Data Table 2] W = [Number of Transfers]
Data Table 1
Data Table 2

Data Table 1	Data Table 2		Data Table 1	Data Table 2
a	i		i	a
b	j		j	b
c	k		k	c
d	l	⟹	l	d
e	m		m	e
f	n		n	f
g	o		o	g
h	p		p	h

Before execution of the XCHG instruction **After execution of the XCHG instruction**

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The contents of MW00000 to MW00009 are exchanged with those of MW00100 MW00109.

XCHG MW00000 ⟹ MW00100 W=00010

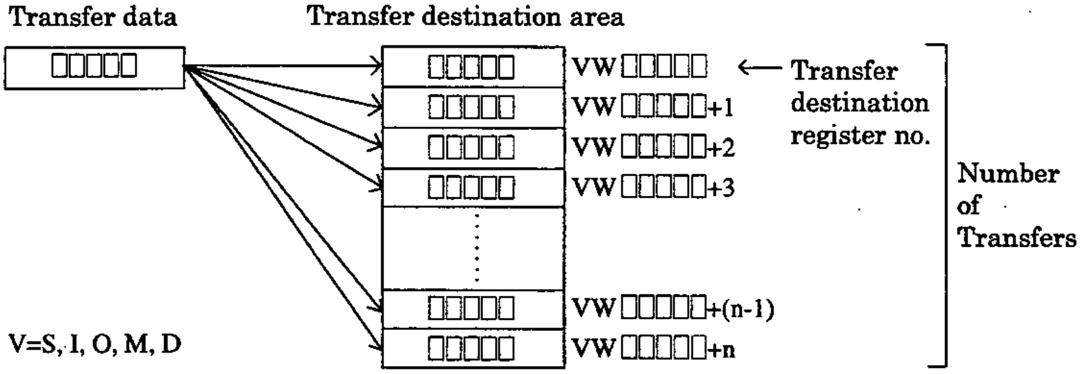
MW00000	1031H	MW00100	2050H		MW00000	2050H	MW00100	1031H
MW00001	1032H	MW00101	2051H		MW00001	2051H	MW00101	1032H
MW00002	1033H	MW00102	2052H		MW00002	2052H	MW00102	1033H
MW00003	1034H	MW00103	2053H	After	MW00003	2053H	MW00103	1034H
MW00004	1035H	MW00104	2054H	transfer ⟶	MW00004	2054H	MW00104	1035H
MW00005	1036H	MW00105	2055H		MW00005	2055H	MW00105	1036H
MW00006	1037H	MW00106	2056H		MW00006	2056H	MW00106	1037H
MW00007	1038H	MW00107	2057H		MW00007	2057H	MW00107	1038H
MW00008	1039H	MW00108	2058H		MW00008	2058H	MW00108	1039H
MW00009	1030H	MW00109	2059H		MW00009	2059H	MW00109	1030H

.9.5 SETW Instruction

[Format]

SETW [Transfer Destination Register No.] [Any integer type register (except for # and C registers)
Any integer type register with subscript (except for # and C registers)] D= [Data to be Transferred] [Any integer type register
Any integer type register with subscript
Constant] W= [Number of Transfers] [Any integer type register
Any integer type register with subscript
Constant]

[Description] The SETW instruction stores the data designated as transfer data in all registers designated by the transfer destination register number and the number of transfers. The storage process is carried out by 1 word in the direction of increasing register number.



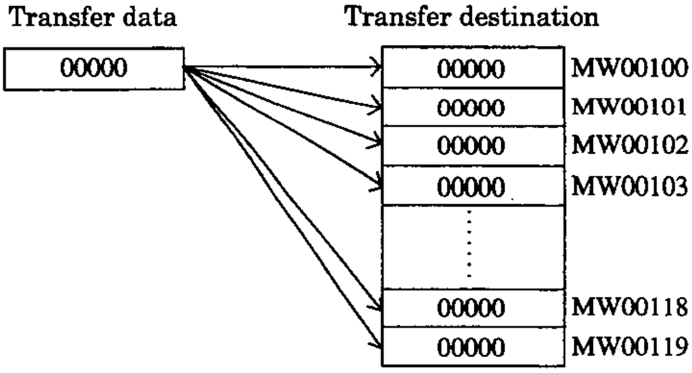
[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The contents of MW00100 to MW00119 are set to 0.

```
SETW    MW00100    D=00000    W=00020
```



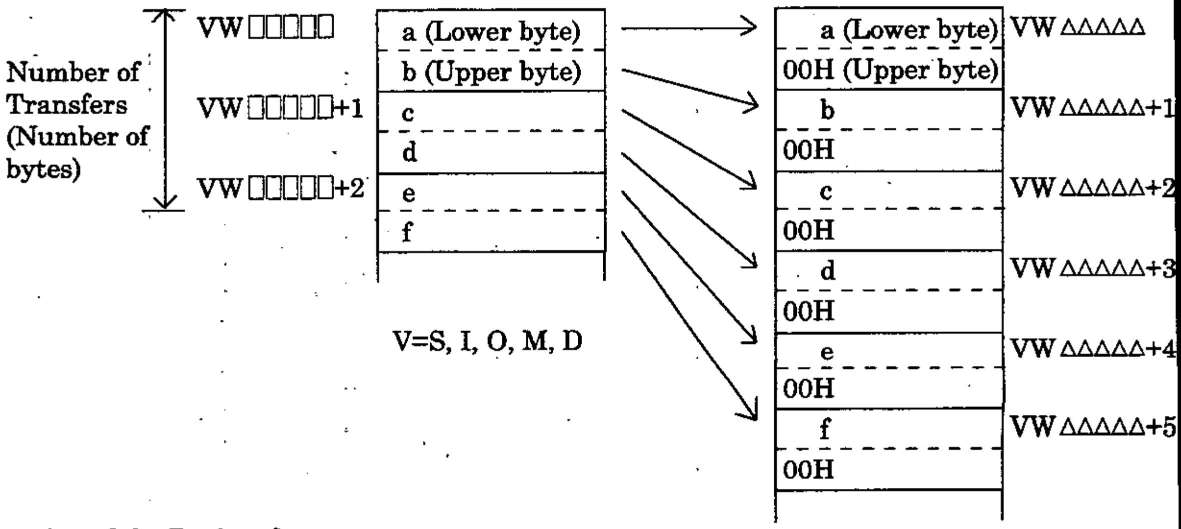
4.9.6 BEXTD Instruction

[Format]

	[Transfer Source Register No.]		[Transfer Destination Register No.]		[Number of Transfers]
BEXTD	Any integer type register Any integer type register with subscript	to	Any integer type register (except for # and C registers) Any integer type register with subscript (except for # and C registers)	B=	Any integer type register Any integer type register with subscript Constant

[Description] The BEXTD instruction stores the byte sequence stored in the transfer source regist area byte by byte in the word sequence of the transfer destination register. The upp byte of the transfer destination register is "0."

In the case of BEXTD VW□□□□□ to VW△△△△△ B=N



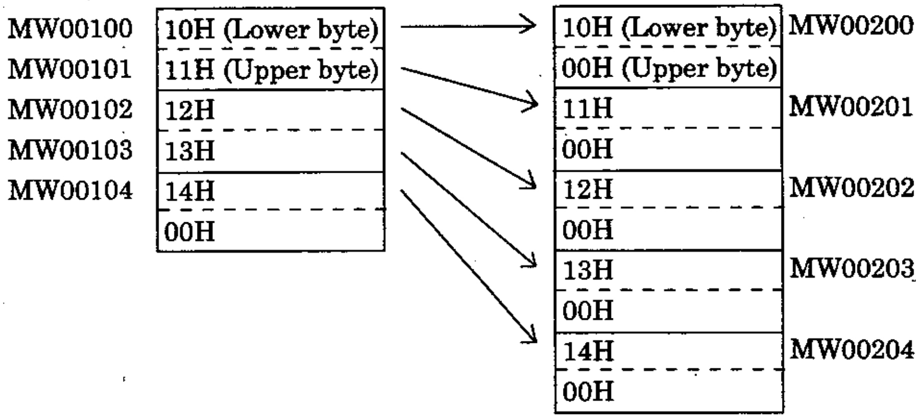
[Operation of the Register]

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The 5 bytes beginning with MW00100 are expanded into five words beginning wit MW00200.

```
BEXTD  MW00100 to MW00200  B=00005
```



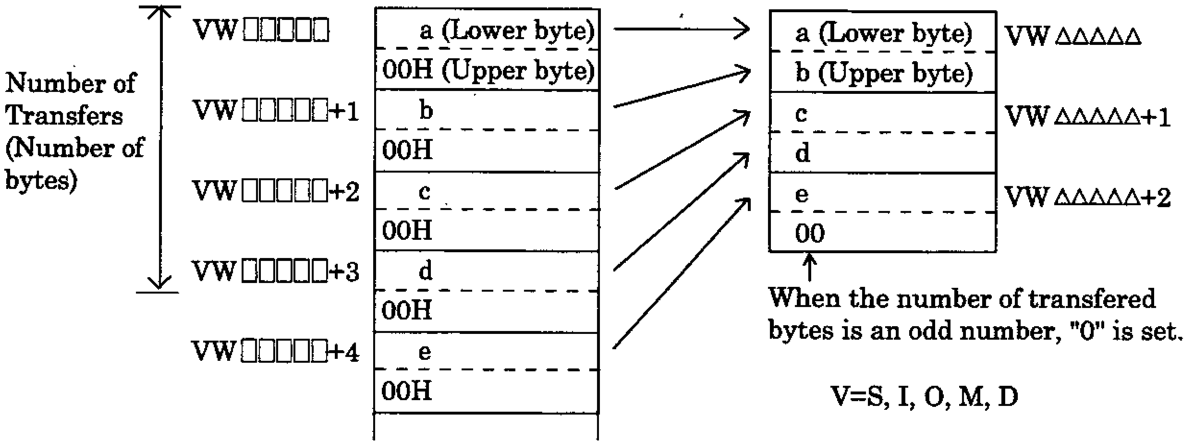
.9.7 BPRESS Instruction

[Format]

	[Transfer Source Register No.]		[Transfer Destination Register No.]		[Number of Transfer bytes]
BPRESS	Any integer type register Any integer type register with subscript	to	Any integer type register (except for # and C registers) Any integer type register with subscript (except for # and C registers)	B=	Any integer type register Any integer type register with subscript Constant

[Description] The BPRESS instruction stores the lower byte of the word sequence stored in the transfer source register area in the byte sequence of the transfer destination register area. The upper byte of the transfer source register is ignored. This is the reverse of the BEXTD instruction.

In the case of BPRESS VW□□□□□ to VW△△△△△ B=N



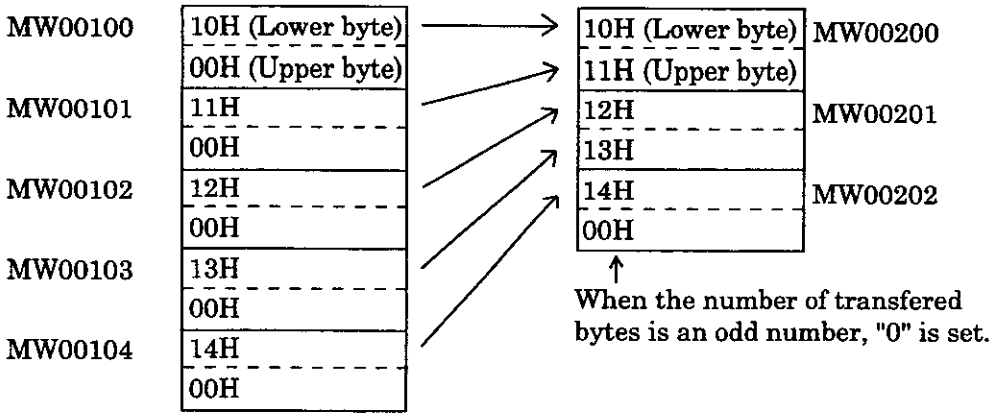
[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The 5 words beginning with MW00100 are compressed into five bytes beginning with MW00200.

```
BPRESS  MW00100  to  MW00200  B=00005
```



4.9.8 BSRCH Instruction

[Format]

	[Head number of the search range]		[Range word number]		[Search data]		[Search result]
BSRCH	Any integer type register Any integer type register with subscript Any double-length integer type register Any double-length integer type register with subscript Any real number type register Any real number type register with subscript	W=	Any integer type register Any integer type register with subscript Constant	D=	Any integer type register Any integer type register with subscript Any double-length integer type register Any double-length integer type register with subscript Any real number type register Any real number type register with subscript Constant	R=	Any integer type regist (except for # and C registers) Any integer type regist with subscript (except f # and C registers)

[Description] The BSRCH instruction uses a binary search method to search for the specified data in the specified search range. The search results (offset number of the search range head register number of matched data) are stored in the specified register.
Before the execution of the BSRCH instruction, it is necessary that the data in the search range be sorted in ascending order. If this is not done, the result will not be correct.
In addition, the result will not be correct if there are two or more identical data.
If no matched data is found, "-1" is stored.

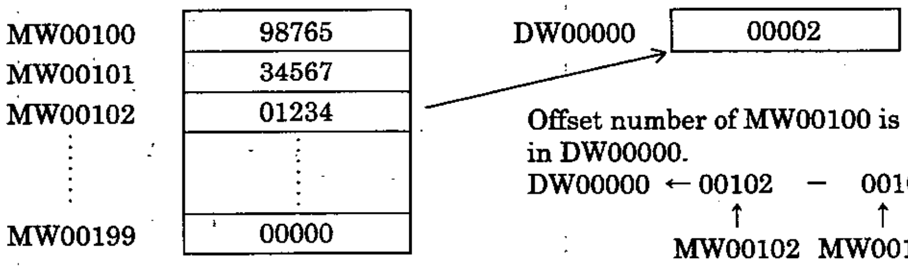
[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Data matching with 01234 are searched for in registers MW00100 to MW00199, and the result is stored in register DW00000.

```
BSRCH   MW00100   W=100   D=01234   R=DW00000
```



Offset number of MW00100 is stored in DW00000.
DW00000 ← 00102 − 00100

9.9 SORT Instruction

[Format]

	[Head number of the sort range]		[Number of range registers]
SORT	Any integer type register (except for # and C registers) Any integer type register with subscript Any double-length integer type register (except for # and C registers) Any double-length integer type register with subscript Any real number type register (except for # and C registers) Any real number type register with subscript	W=	Any integer type register Any integer type register with subscript Any double-length integer type register Any double-length integer type register with subscript Any real number type register Any real number type register with subscript

[Description] The SORT instruction arranges data in the specified register range in ascending order.

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○ : stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The data in registers MW00100 to MW00199 are sorted in ascending order.

```
    SORT    MW00100    W=00020
```

4.9.10 SHFTL Instruction and SHFTR Instruction

[Format]

	[Head Bit Address]		[Number of Shifts]		[Bit Width]
SHFTL SHFTR	Any bit type register (except for # and C registers) Any bit type register with subscript (except for # and C registers)	N=	Any integer type register Any integer type register with subscript Constant	W=	Any integer type registe Any integer type registe with subscript Constant

[Description] The SHIFTL (SHIFTR) instruction shifts to the left (right) by only the specified num of shifts the bit sequence specified by head bit address and bit width.
As shown in Fig. 4.6, bit data that overflows the bit width is thrown away, and insuffici bits become 0.

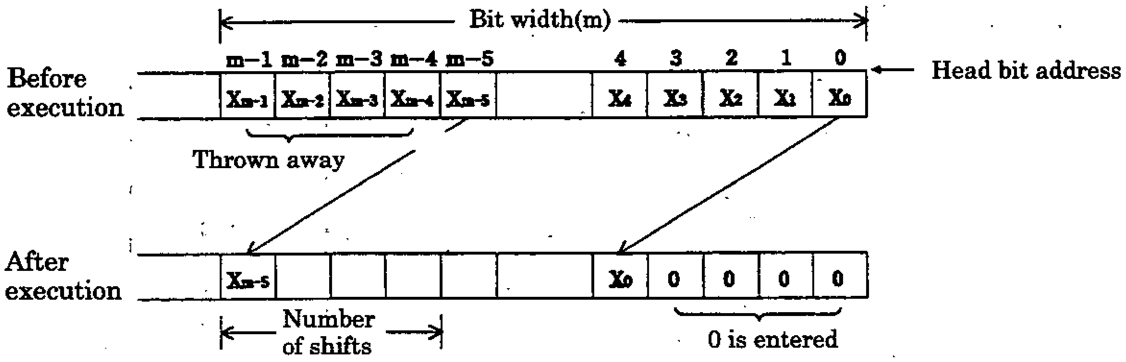


Fig. 4.6 The SHIFT Operation

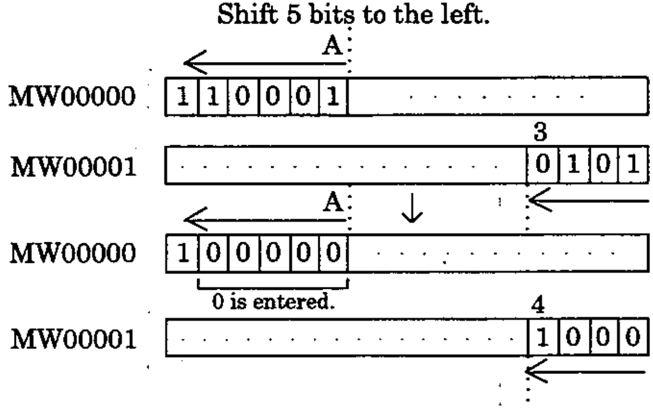
[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] (1) SHFTL A ten-bit wide section of data with MB00000A (bit A of MW00000) as t head is shifted five bits to the left.

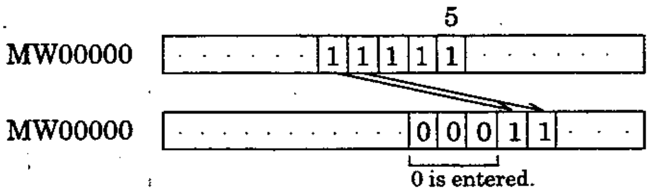
```
SHFTL MB00000A N=5 W=10
```



Note: The upper five bits are thrown away.

(2) SHFTR A five-bit wide section of data with MB00005 (bit 5 of MW00000) as t head is shifted three bits to the right.

```
SHFTR MB000005 N=3 W=5
```



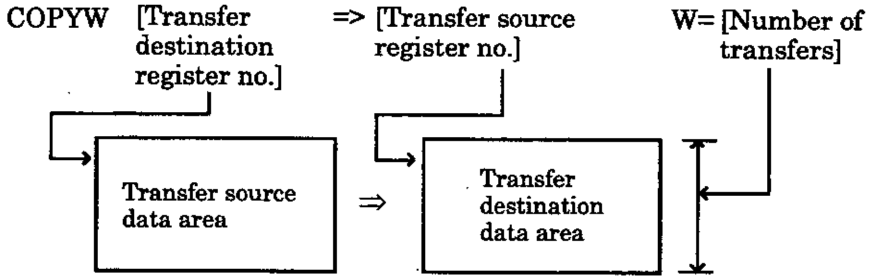
Note: The lower three bi are thrown away.

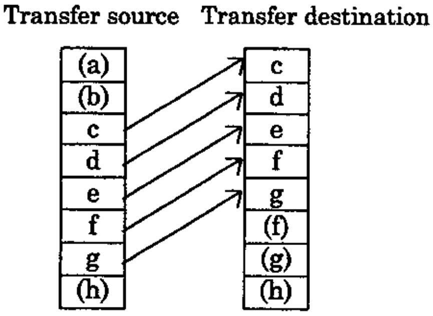
.9.11 COPYW Instruction

[Format]

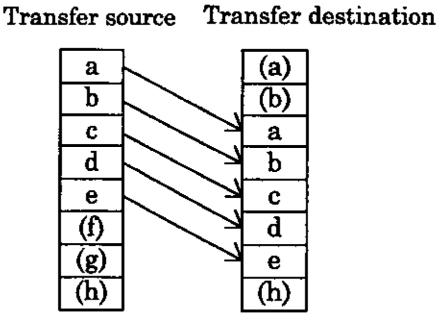
	[Transfer Source Register No.]		[Transfer Destination Register No.]		[Number of Transfers]
COPYW	Any bit type register Any bit type register with subscript	N=	Any bit type register (except for # and C registers) Any bit type register with subscript (except for # and C registers)	W=	Any integer type register Any integer type register with subscript Constant

[Description] The COPYW instruction transfers the specified number of word data to the head of the transfer destination register from the head of the transfer source register. The transfer operation copies the data in a block from the transfer source to the transfer destination. Even if there is overlap between the transfer source and the transfer destination, the full transfer data block is copied to the transfer destination.


COPYW
[Transfer destination register no.]
=> [Transfer source register no.]
W= [Number of transfers]
Transfer source data area
⇒
Transfer destination data area



When the transfer source and transfer destination overlap (1)



When the transfer source and transfer destination overlap (2)

[Operation of the Register]

A	F	B	I	J
○	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Examples(s)] The word data of MW00000 to MW00009 are transferred to MW00100 to MW00109.

```
COPYW  MW00000 => MW00100  W=00010
```

MW00000	1032H		MW00100	1032H
MW00001	1133H	After transfer →	MW00101	1133H
MW00002	1234H		MW00102	1234H
	⋮			⋮
MW00008	1841H		MW00108	1841H
MW00009	1842H		MW00109	1842H

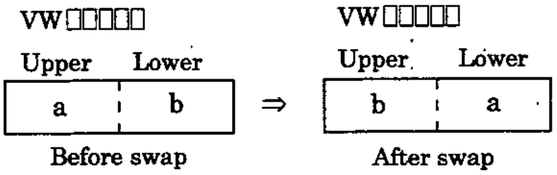
4.9.12 BSWAP Instruction

[Format] [Target register number]

BSWAP [Any bit type register
(except for # and C registers)
Any bit type register with subscript
(except for # and C registers)]

[Description] The BSWAP instruction swaps the upper and lower bytes of the specified register.

(Target register)
In the case of BSWAP VW□□□□□



V=S, I, O, M, D

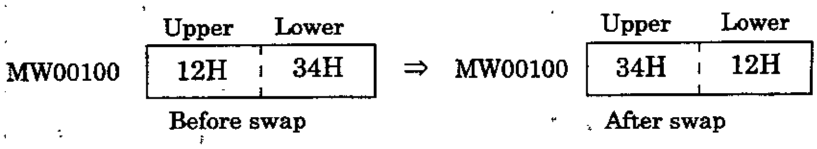
[Operation of the Register]

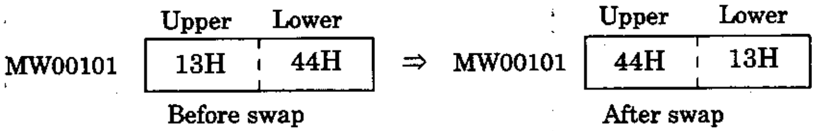
A	F	B	I	J
○	○	○	○	○

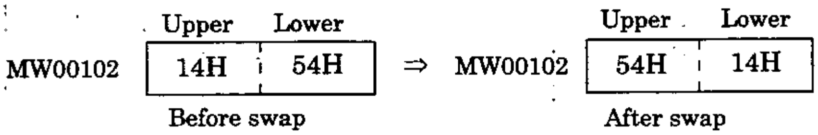
○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The upper and lower bytes of MW00100 to MW00102 are swapped.

```
FOR I = 00000 to 00002 by 00001
BSWAP MW00100 i
FEND
```







.10 Basic Function Instructions

.10.1 SQRT Instruction

[Format] SQRT

[Description] This instruction leaves the square root of integer type or real number type data as the operation result. The input unit and the output result will differ according to whether the data are of an integer type or a real number type. This instruction cannot be used for double-length integer type data.

Integer Type Data

The operation result will differ slightly from the square root in mathematical terms. To be more precise, the operation result is expressed by the following formula:

$32768 \times \text{sign(A)} \times \text{SQRT}(|A|/32768)$

sign (A) : sign of register A

$|A|$: absolute value of register A

That is, the operation result will be equal to the mathematical square root multiplied by $128\sqrt{2}$ (approx. 181.02). When the input is a negative number, the square root of the absolute value is determined and the negative of this square root is left as the operation result in the A register.

The maximum operation error of the output value is ± 2.

Real Number Type Data

The immediately preceding operation result (F register) is used as the input and the square root thereof is left in the F register. When the input is a negative number the square root of the absolute value is determined and the negative of this square root is left as the operation result in the A register. This instruction can be used inside a real number type operation.

[Operation of the Register]

Integer type data

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

Real number type data

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data

When the input is a positive number

```
├MW00100   SQRT                    ⟹ MW00102
 (00064)                               (01448)
```

When the input is a negative number

```
├MW00100   SQRT                    ⟹ MW00102
 ( -00064)                             ( -01448)
```

Real number type data

When the input is a positive number

```
╟DF00200   SQRT                    ⟹ DF00202
 (64.0)                                (8.0)
```

When the input is a negative number

```
╟DF00200   SQRT                    ⟹ DF00202
 ( -64.0)                              ( -8.0)
```

4.10.2 SIN Instruction

[Format] SIN

[Description] This instruction leaves the sine of integer type or real number type data as the operati result. The input unit and the output result will differ according to whether the da are of an integer type or a real number type. This instruction cannot be used for doub length integer type data.

Integer Type Data

This instruction can be used in the range -327.68 ~ 327.67 degrees. T immediately preceding operation result (A register) is used as the input (1 = 0. degrees) and the operation result is left in the A register.
Upon output, the operation result is multiplied by 10000.
If a number outside the range -327.68 to 327.67 is mistakenly entered, a corr result will not be obtained. For example, if 360.00 is entered, a result of -295. degrees is output.

Real Number Type Data

The immediately preceding operation result (F register) is used as the input (u = degrees) and the sine thereof is left in the F register. This instruction can used inside a real number type operation.

[Operation of the Register]

Integer type data

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

Real number type data

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data

```
├MW00100   SIN                    ⟹ MW00102
  (03000)                             (05000)
```

Input θ = 30 degrees (MW00100 = 30 × 100 = 3000)
Output SIN(θ) = 0.50 (MW00102 = 0.50 × 10000 = 5000)

Real number type data

```
||├DF00200   SIN                  ⟹ DF00202
   (30.0)                             (0.5)
```

.10.3 COS Instruction

[Format] COS

[Description] This instruction leaves the cosine of integer type or real number type data as the operation result. The input unit and the output result will differ according to whether the data are of an integer type or a real number type. This instruction cannot be used for double-length integer type data.

Integer Type Data

This instruction can be used in the range -327.68 ~ 327.67 degrees. The immediately preceding operation result (A register) is used as the input (1 = 0.01 degrees) and the operation result is left in the A register.
Upon output, the operation result is multiplied by 10000.
If a number outside the range -327.68 to 327.67 is mistakenly entered, a correct result will not be obtained. For example, if 360.00 is entered, a result of -295.36 degrees is output.

Real Number Type Data

The immediately preceding operation result (F register) is used as the input (unit = degrees) and the cosine thereof is left in the F register. This instruction can be used inside a real number type operation.

[Operation of the Register]

Integer type data

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

Real number type data

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data

```
├ MW00100   COS                  ⟹ MW00102
  (06000)                           (05000)
```

Input θ = 60 degrees (MW00100 = 60 × 100 = 6000)
Output COS(θ) = 0.50 (MW00102 = 0.50 × 10000 = 5000)

Real number type data

```
||├ DF00200   COS                ⟹ DF00202
    (60.0)                           (0.5)
```

4.10.4 TAN Instruction

[Format] TAN

[Description] With the TAN instruction, the immediately preceding operation result (F register) used as the input (unit = degrees) and the tangent thereof is left in the F register. Th instruction can be used inside a real number type operation.

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The tangent of the input value (θ = 45.0 degrees) [TAN(θ) = 1.0] is calculated.

```
|├ DF00200  TAN                          ⇒DF00202
   (45.0)                                  (1.0)
```

4.10.5 ASIN instruction

[Format] ASIN

[Description] With the ASIN instruction, the immediately preceding operation result (F register) used as the input (unit = degrees) and the arc sine thereof is left in the F register. Th instruction can be used inside a real number type operation.

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The arc sine of the input value (θ = 0.5) [ASIN(0.5) = θ = 30.0 degrees] is calculate

```
|├ DF00200
   (0.5)
 ASIN                                ⇒DF00202
                                       (30.0)
```

4.10.6 ACOS Instruction

[Format] ACOS

[Description] With the ACOS instruction, the immediately preceding operation result (F register) used as the input (unit = degrees) and the arc cosine thereof is left in the F registe This instruction can be used inside a real number type operation.

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The arc cosine of the input value (θ = 0.5) [ACOS(0.5) = θ = 60.0 degrees] is calculate

```
|├ DF00200
   (0.5)
 ACOS                                ⇒DF00202
                                       (60.0)
```

10.7 ATAN Instruction

[Format] ATAN

[Description] This instruction leaves the arc tangent of integer type or real number type data as the operation result. The input unit and the output result will differ according to whether the data are of an integer type or a real number type. This instruction cannot be used for double-length integer type data.

Integer Type Data

This instruction can be used in the range -327.68 to 327.67. The immediately preceding operation result (A register) is used as the input (1 = 0.01) and the operation result is left in the A register.
Upon output, the operation result is multiplied by 100 degrees.

Real Number Type Data

The immediately preceding operation result (F register) is used as the input and the arc tangent thereof (unit = degrees) is left in the F register. This instruction can be used inside a real number type operation.

[Operation of the Register]

Integer type data

A	F	B	I	J
×	○	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

Real number type data

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer type data

```
├ MW00100
  (00100)
ATAN                          ⇒MW00102
                                (04500)
```

Input X = 1.00 (MW00100 = 1.00 × 100 = 100)
Output θ = 45 degrees (MW00102 = 45 × 100 = 4500)

Real number type data

```
|├ DF00200
  (1.0)
ATAN                          ⇒DF00202
                                (45.0)
```

4.10.8 EXP Instruction

[Format] EXP

[Description] With the EXP instruction, the immediately preceding operation result (F register) used as the input (x) and the natural logarithmic base (e) to the power of the inp value (e^x) is left in the F register as the operation result. This instruction can be us only inside a real number type operation.

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] e (= 2.7183) to the power of the input value (x = 1.0) is calculated.

```
||- DF00200   EXP                   ==> DF00202
    (1.0)                               (2.7183)
```

4.10.9 LN Instruction

[Format] LN

[Description] With the LN instruction, the immediately preceding operation result (F register) used as the input (x) and the natural logarithm (Log$_e^x$)thereof is left in the F register the operation result. This instruction can be used inside a real number type operatio

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Calculate the natural logarithm of the input value (x = 10.0) [Log$_e$(x) = 2.3026].

```
||- DF00200   LN                    ==> DF00202
    (10.0)                              (2.3026)
```

4.10.10 LOG Instruction

[Format] LOG

[Description] With the LOG instruction, the immediately preceding operation result (F register) used as the input (x) and the common logarithm (log$_{10}^x$) thereof is left in the F regist as the operation result. This instruction can be used inside a real number type operatio

[Operation of the Register]

A	F	B	I	J
○	×	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The common logarithm of the input value (x = 10.0) [Log$_{10}$(x) = 1.0] is calculated.

```
||- DF00200   LOG                   ==> DF00202
    (10.0)                              (1.0)
```

.11 DDC Instructions

.11.1 DZA Instruction

[Format] [Designated Dead Zone Value]

DZA
- Any integer type register
- Any integer type register with subscript
- Any double-length integer type register
- Any double-length integer type register with subscript
- Any real number type register
- Any real number type register with subscript
- Subscript register
- Constant

[Description] The DZA instruction executes a dead zone operation on integer, double-length integer, or real number type data. Where X is the input value, D is the designated dead zone value, and Y is the output value, the following operation is performed:

(a) $Y = X\ (|X| \geqq |D|)$
(b) $Y = 0\ (|X| < |D|)$

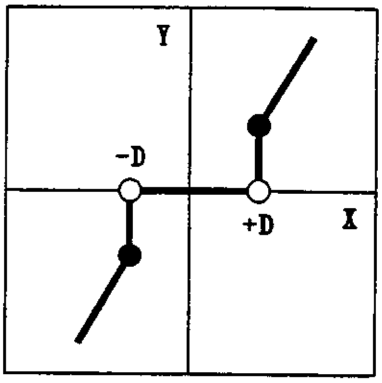


Fig. 4.7 Operation of the DZA Instruction

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊩ .

[Example(s)] Integer type operation

```
⊢MW00100
  (00150)
  (00050)
DZA  00100                            ⇒MW00102
                                        (00150)   ← Outside dead zone
                                        (00000)   ← Within dead zone
```

Double-length integer type operation

```
⊢ML00100
  (200000)
  (050000)
DZA  100000                           ⇒ML00102
                                        (200000)  ← Outside dead zone
                                        (000000)  ← Within dead zone
```

Real number type operation

```
╟ DF00200
   (150.0)
   (50.0)
DZA  100.0                         ⇒DF00202
                                     (150.0)    <— Outside dead zon
                                     (0.0)      <— Within dead zone
```

4.11.2 DZB Instruction

[Format]

[Designated Dead Zone Value]

DZB
- Any integer type register
- Any integer type register with subscript
- Any double-length integer type register
- Any double-length integer type register with subscript
- Any real number type register
- Any real number type register with subscript
- Subscript register
- Constant

[Description] The DZB instruction executes a dead zone operation on integer, double-length integ or real number type data. Where X is the input value, D is the designated dead zo value, and Y is the output value, the following operation is performed:

(a) $Y = X - |D|\ (|X| \geqq |D|, X \geqq 0)$

(b) $Y = X + |D|\ (|X| \geqq |D|, X \leqq 0)$

(c) $Y = 0\ (|X| < |D|)$

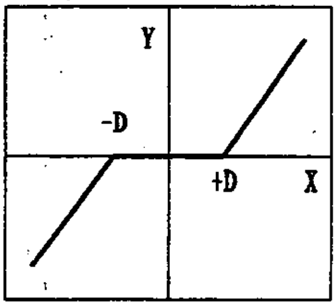


Fig. 4.8 Operation of the DZB Instruction

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ├ . Will be stored if the operation does not start with a ├ .

*2: Will not be stored if the operation starts with a ╟ . Will be stored if the operation does not start with a ╟ .

[Example(s)]

Integer type operation

```
|- MW00100
   (00150)
   (00050)
DZB  00100                        ⇒ MW00102
                                    (00050)   ← Outside dead zone
                                    (00000)   ← Within dead zone
```

Double-length integer type operation

```
|- ML00100
   (200000)
   (050000)
DZB  100000                       ⇒ ML00102
                                    (100000)  ← Outside dead zone
                                    (000000)  ← Within dead zone
```

Real number type operation

```
||- DF00200
   (150.0)
   (50.0)
DZB  100.0                        ⇒ DF00202
                                    (50.0)    ← Outside dead zone
                                    (0.0)     ← Within dead zone
```

4.11.3 LIMIT Instruction

[Format]

LIMIT

[Lower Limit]	[Upper Limit]
Any integer type register	Any integer type register
Any integer type register with subscript	Any integer type register with subscript
Any double-length integer type register	Any double-length integer type register
Any double-length integer type register with subscript	Any double-length integer type register wi subscript
Any real number type register	Any real number type register
Any real number type register with subscript	Any real number type register with subscript
Subscript register	Subscript register
Constant	Constant

[Description] The LIMIT instruction executes an upper/lower limit operation on integer, double-leng integer, or real number type data. The following operation is performed:

(a) $Y = A \ (X < A)$
(b) $Y = X \ (A \leqq X \leqq B)$
(c) $Y = B \ (B < X)$

Where X is the inout value. A is the lower limit, B is the upper limit, and Y is the outp

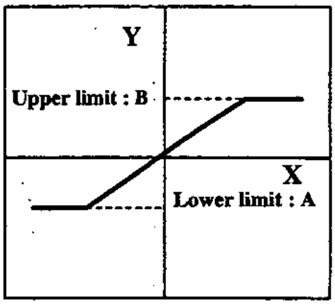


Fig. 4.9 Operation of the LIMIT Instruction

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2: Will not be stored if the operation starts with a ‖⊢ . Will be stored if the operation does not start with a ‖⊢ .

[Example(s)] **■ Integer type operation**

```
⊢MW00100
LIMIT -00100  00100                    ⇒MW00102
```

Input (MW00100)	Output (MW0010)
-100>MW00100	-00100 (under the lower limit)
-100≦MW00100≦100	Value of MW00100 (within the upper and lower limits)
MW00100>100	00100 (above the upper limit)

■ Double-length integer type operation

```
⊢ML00100
LIMIT -100000  100000                  ⇒ML00102
```

Input (ML00100)	Output (ML00102)
-100000>ML00100	-100000 (under the lower limit)
-100000≦ML00100≦100000	Value of ML00100 (within the upper and lower limits)
ML00100>100000	100000 (above the upper limit)

■ Real number type operation

```
|⊢ MF00200
LIMIT  -100.0   100.0                    ⇒MF00202
```

Input (MF00200)	Output (MF00202)
-100.0>DF00100	-100.0 (under the lower limit)
-100.0≦DF00100≦100.0	Value of MF00200 (within the upper and lower limits)
DF00100>100.0	100.0 (above the upper limit)

4.11.4 PI Instruction

[Format] [Head Address of Parameter Table]

PI [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The PI instruction executes a PI operation in accordance with the contents of parameter table that is set in advance. The input (X) to the PI operation must be a integer type or real number type value. The configuration of the parameter table wil differ according to whether the parameters are of an integer type or of a real numbe type. Double-length integer type parameters cannot be used (operations will b performed with each parameter being handled as an integer consisting of the lower 1 bits).

Table 4.17 Table of Integer Type PI Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	Kp	P gain	Gain of the P correction (a gain of 1 is set to 100)	IN
2	W	Ki	Integration adjustment gain	Gain of the integration circuit input (a gain of 1 is set to 100)	IN
3	W	Ti	Integration time	Integration time (ms)	IN
4	W	IUL	Upper integration limit	Upper limit for the I correction value	IN
5	W	ILL	Lower integration limit	Lower limit for the I correction value	IN
6	W	UL	Upper PI limit	Upper limit for the P+I correction value	IN
7	W	LL	Lower PI limit	Lower limit for the P+I correction value	IN
8	W	DB	PI output dead band	Width of the dead band for the P+I correction value	IN
9	W	Y	PI output	PI correction output (also output to the A register)	OUT
10	W	Yi	I correction value	Storage of the I correction value	OUT
11	W	IREM	I remainder	Storage of the I remainder	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	Integration reset	"ON" is input when integration is reset.	IN
1 to 7	———	(Reserve)	Reserve relay for input	IN
8 to F	———	(Reserve)	Reserve relay for output	OUT

Table 4.18 Table of Real Type PI Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	———	(Reserve)	Reserve register	———
2	F	Kp	P gain	Gain of the P correction	IN
4	F	Ki	Integration adjustment gain	Gain of the integration circuit input	IN
6	F	Ti	Integration time	Integration time (s)	IN
8	F	IUL	Upper integration limit	Upper limit for the I correction value	IN
10	F	ILL	Lower integration limit	Lower limit for the I correction value	IN
12	F	UL	Upper PI limit	Upper limit for the P+I correction value	IN
14	F	LL	Lower PI limit	Lower limit for the P+I correction value	IN
16	F	DB	PI output dead band	Width of the dead band for the P+I correction value	IN
18	F	Y	PI output	PI correction output (also output to the A register)	OUT
20	F	Yi	I correction value	Storage of the I correction value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	Integration reset	"ON" is input when integration is reset.	IN
1 to 7	———	(Reserve)	Reserve relay for input	IN
8 to F	———	(Reserve)	Reserve relay for output	OUT

Here, the PI operation is expressed as follows:

$$\frac{Y}{X} = Kp + Ki \times \frac{1}{Ti \times S}$$

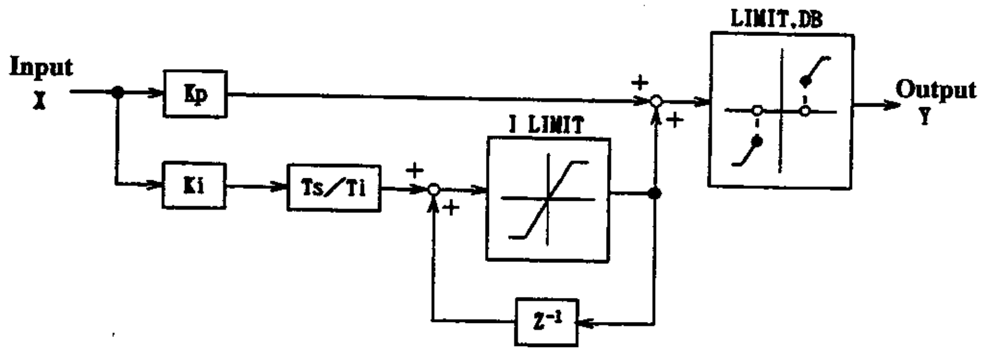
X: deviation input value
Y: output value

The following operation is performed within the PI instruction:

$$Y = Kp \times X + \{(Ki \times X + IREM) / \frac{Ti}{Ts} + Yi'\}$$

Yi' : previous I output value Ts : scan time set value

Block Diagram



When the P+I correction value reaches the upper or lower PI limit (UL, LL) or the PI dead band (DB)

When the present P correction value and the I correction value are the same in sign (diverging), the I correction value is not renewed but is kept at the previous value. Oppositely, if the P and I correction values are opposite in sign (converging towards 0), the I correction value is renewed by the present value.

When the integration reset (IRST) is "ON"

Yi = 0 and IREM = 0 are output

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2: Will not be stored if the operation starts with a ⊩ . Will be stored if the operation does not start with a ⊩ .

[Example(s)] Integer type operation
MW00100 to MW00111 are used for the parameter table.

```
├ MW00010   ←Deviation input value
  PI  MA00100                          ⇒MW00011
      ↑                                    ↑
  Head address of                     PI output value
  parameter table
```

Real number type operation
MF00200 to MF00220 are used for the parameter table.

```
╟ MF00200   ←Deviation input value
  PI  MA00200                          ⇒MF00022
      ↑                                    ↑
  Head address of                     PI output value
  parameter table
```

4.11.5 PD Instruction

[Format]

[Head Address of Parameter Table]

PD [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The PD instruction executes a PD operation in accordance with the contents of a parameter table that is set in advance. The input (X) to the PD operation must be an integer type or real number type value. The configuration of the parameter table will differ according to whether the parameters are of an integer type or of a real number type. Double-length integer type parameters cannot be used (operations will be performed with each parameter being handled as an integer consisting of the lower 16 bits).

Table 4.19 Table of Integer Type PD Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	Kp	P gain	Gain of the P correction (a gain of 1 is set to 100)	IN
2	W	Kd	D gain	Gain of the differentiation circuit input (a gain of 1 is set to 100)	IN
3	W	Td1	Divergence differentiation time	The differentiation time (ms) used in the case of diverging input.	IN
4	W	Td2	Convergence differentiation time	The differentiation time (ms) used in the case of converging input.	IN
5	W	UL	Upper PD limit	Upper limit for the P+D correction value	IN
6	W	LL	Lower PD limit	Lower limit for the P+D correction value	IN
7	W	DB	PD output dead band	Width of the dead band for the P+D correction value	IN
8	W	Y	PD output	PD correction output (also output to the A register)	OUT
9	W	X	Input value storage	Storage of the present deviation input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Table 4.20 Table of Real Type PD Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	——	(Reserve)	Reserve register	——
2	F	Kp	P gain	Gain of the P correction	IN
4	F	Kd	D gain	Gain of the differentiation circuit input	IN
6	F	Td1	Divergence differentiation time	The differentiation time (s) used in the case of diverging input.	IN
8	F	Td2	Convergence differentiation time	The differentiation time (s) used in the case of converging input.	IN
10	F	UL	Upper PD limit	Upper limit for the P+D correction value	IN
12	F	LL	Lower PD limit	Lower limit for the P+D correction value	IN
14	F	DB	PD output dead band	Width of the dead band for the P+D correction value	IN
16	F	Y	PD output	PD correction output (also output to the A register)	OUT
18	F	X	Input value storage	Storage of the present deviation input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Here, the PD operation is expressed as follows:

$$\frac{Y}{X} = Kp + Kd \times Td \times S$$

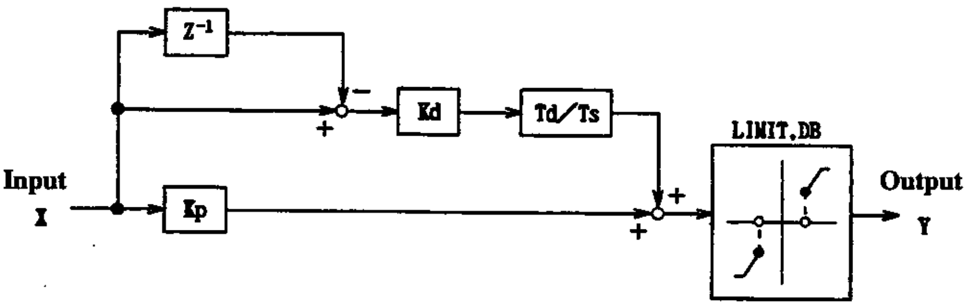
X: deviation input value   Y: output value

The following operation is performed within the PD instruction:

$$Y = Kp \times X + Kd \times (X - X') \times \frac{Td}{Ts}$$

Xi' : previous input value   Ts : scan time set value

Block Diagram



When the change in deviation output (X-X') and the previous deviation input (X') are the same in sign (diverging) in the differentiation (D) operation

The divergence differentiation time (Td1) is used as the differentiation time.

When the change in deviation output (X-X') and the previous deviation input (X') are opposite in sign (converging) in the differentiation (D) operation

The convergence differentiation time (Td2) is used as the differentiation time.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢. Will be stored if the operation does not start with a ⊢.

*2: Will not be stored if the operation starts with a ⊩. Will be stored if the operation does not start with a ⊩.

[Example(s)] Integer type operation

MW00100 to MW00109 are used for the parameter table.

```
⊢ MW00010   ← Deviation input value
  PD MA00100                          ⇒ MW00011
```

Head address of parameter table

PD output value

Real number type operation

MF00200 to MF00218 are used for the parameter table.

```
⊩ MF00200   ← Deviation input value
  PD MA00200                          ⇒ MF00022
```

Head address of parameter table

PD output value

4.11.6 PID Instruction

[Format] [Head Address of Parameter Table]

PID [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The PID instruction executes a PID operation in accordance with the contents of a parameter table that is set in advance. The input (X) to the PID operation must be an integer type or real number type value. The configuration of the parameter table wil differ according to whether the parameters are of an integer type or of a real numbe type. Double-length integer type parameters cannot be used (operations will b performed with each parameter being handled as an integer consisting of the lower 1 bits).

Table 4.21 Table of Integer Type PID Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output*1	IN/OUT
1	W	Kp	P gain	Gain of the P correction (a gain of 1 is set to 100)	IN
2	W	Ki	I gain	Gain of the integration circuit input (a gain of 1 is set to 100)	IN
3	W	Kd	D gain	Gain of the differentiation circuit input (a gain of 1 is set to 100)	IN
4	W	Ti	Integration time	Integration time (ms)	IN
5	W	Td1	Divergence differentiation time	The differentiation time (ms) used in the case of diverging input.	IN
6	W	Td2	Convergence differentiation time	The differentiation time (ms) used in the case of converging input.	IN
7	W	IUL	Upper integration limit	Upper limit for the I correction value	IN
8	W	ILL	Lower integration limit	Lower limit for the I correction value	IN
9	W	UL	Upper PID limit	Upper limit for the P+I+D correction value	IN
10	W	LL	Lower PID limit	Lower limit for the P+I+D correction value	IN
11	W	DB	PID output dead band	Width of the dead band for the P+I+D correction value	IN
12	W	Y	PID output	PID correction output (also output to the A register)	OUT
13	W	Yi	I correction value	Storage of the I correction value	OUT
14	W	IREM	I remainder	Storage of the I remainder	OUT
15	W	X	Input value storage	Storage of the present deviation input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	Integration reset	"ON" is input when integration is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Table 4.22 Table of Real Type PID Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	—	(Reserve)	Reserve register	—
2	F	Kp	P gain	Gain of the P correction	IN
4	F	Ki	I gain	Gain of the integration circuit input	IN
6	F	Kd	D gain	Gain of the differentiation circuit input	IN
8	F	Ti	Integration time	Integration time (s)	IN
10	F	Td1	Divergence differentiation time	The differentiation time (s) used in the case of diverging input.	IN
12	F	Td2	Convergence differentiation time	The differentiation time (s) used in the case of converging input.	IN
14	F	IUL	Upper integration limit	Upper limit for the I correction value	IN
16	F	ILL	Lower integration limit	Lower limit for the I correction value	IN
18	F	UL	Upper PID limit	Upper limit for the P+I+D correction value	IN
20	F	LL	Lower PID limit	Lower limit for the P+I+D correction value	IN
22	F	DB	PID output dead band	Width of the dead band for the P+I+D correction value	IN
24	F	Y	PID output	PID correction output (also output to the A register)	OUT
26	F	Yi	I correction value	Storage of the I correction value	OUT
28	F	X	Input value storage	Storage of the present deviation input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	Integration reset	"ON" is input when integration is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Here, the PID operation is expressed as follows:

$$\frac{Y}{X} = Kp + Ki \times \frac{1}{Ti \times S} + Kd \times Td \times S$$

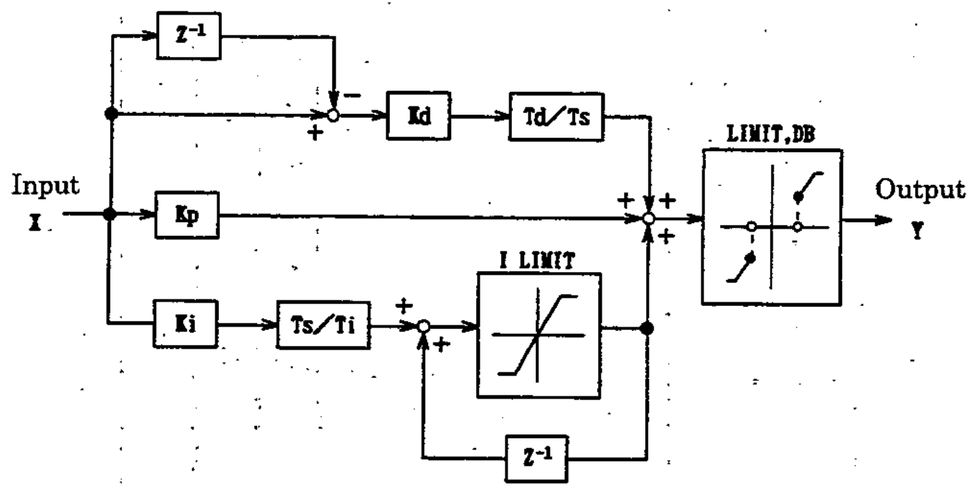
X: deviation input value
Y: output value

The following operation is performed within the PID instruction:

$$Y = Kp \times X + \{(Ki \times X + IREM) / \frac{Ti}{Ts} + Yi'\} + Kd \times (X\text{-}X') \times \frac{Td}{Ts}$$

X' : previous input value
Yi' : previous I output value
Ts : scan time set value

Block Diagram



When the P+I+D correction value reaches the upper or lower PID limit (UL, LL) or the PID dead band (DB)

When the present P correction value and the I correction value are the same in sign (diverging), the I correction value is not renewed but is kept at the previous value. Oppositely, if the P and I correction values are opposite in sign (converging towards 0), the I correction value is renewed with the present value.

When the change in deviation output (X-X') and the previous deviation input (X') are the same in sign (diverging) in the differentiation (D) operation

The divergence differentiation time (Td1) is used as the differentiation time.

When the change in deviation output (X-X') and the previous deviation input (X') are opposite in sign (converging) in the differentiation (D) operation

The convergence differentiation time (Td2) is used as the differentiation time.

When the integration reset (IRST) is "ON"

Yi = 0 and IREM = 0 are output.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ├ . Will be stored if the operation does not start with a ├ .

*2: Will not be stored if the operation starts with a ╟ . Will be stored if the operation does not start with a ╟ .

[Example(s)] Integer type operation

MW00100 to MW00115 are used for the parameter table.

```
├ MW00010  ←— Deviation input value
PID  MA00100                          ⇒MW00011
      ↑                                   ↑
 Head address of                   PID output value
 parameter table
```

Real number type operation

MF00200 to MF00228 are used for the parameter table.

```
╟MF00200  ←— Deviation input value
PID  MA00200                          ⇒MF00022
      ↑                                   ↑
 Head address of                   PID output value
 parameter table
```

.11.7 LAG Instruction

[Format]

[Head Address of Parameter Table]
LAG Register address (except for # and C registers)
Register address with subscript (except for # and C registers)

[Description] The LAG instruction computes the first-order lag in accordance with the contents of a parameter table that is set in advance. The input (X) to the LAG operation must be an integer type or real number type value. The configuration of the parameter table will differ according to whether the parameters are of an integer type or of a real number type. Double-length integer type parameters cannot be used (operations will be performed with each parameter being handled as an integer consisting of the lower 16 bits).

Table 4.23 Table of Integer Type LAG Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *S[1]	IN/OUT
1	W	T	First-order lag time constant	First-order lag time constant (ms)	IN
2	W	Y	LAG output	LAG output (also output to the A register)	OUT
3	W	REM	Remainder	Storage of remainder	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	LAG reset	"ON" is input when LAG is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Table 4.24 Table of Real Type LAG Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *[1]	IN/OUT
1	W	——	(Reserve)	Reserve register	——
2	F	T	First-order lag time constant	First-order lag time constant (s)	IN
4	F	Y	LAG output	LAG output (also output to the F register)	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	LAG reset	"ON" is input when LAG is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Here, the LAG operation is expressed as follows:

$$\frac{Y}{X} = \frac{1}{1 + T \times S} \quad ;\text{ie. } T \times (dY/dt) + Y = X$$

The following operation is performed within the LAG instruction with dt=Ts and dY=Y-Y':

$$Y = \frac{T \times Y' + Ts \times X + REM}{T + Ts}$$

X : input value

Y : output value

Y': previous output value

Ts : scan time set value

Y=0 and REM=0 are output when the LAG reset (RST) is "ON".

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ├ . Will be stored if the operation does not start with a ├ .
*2: Will not be stored if the operation starts with a ╟ . Will be stored if the operation does not start with a ╟ .

[Example(s)] Integer type operation
MW00100 to MW00103 are used for the parameter table.

```
├ MW00010  ←Input value
LAG  MA00100                      ⇒MW00011
       ↑                               ↑
Head address of parameter table   LAG output value
```

Real number type operation
MF00200 to MF00204 are used for the parameter table.

```
╟ MF00200  ←Input value
LAG  MA00200                      ⇒MF00022
       ↑                               ↑
Head address of parameter table   LAG output value
```

4.11.8 LLAG Instruction

[Format] [Head Address of Parameter Table]

LLAG [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The LLAG instruction computes the phase lead/lag in accordance with the contents of parameter table that is set in advance. The input (X) to the LLAG operation must be a integer type or real number type value. The configuration of the parameter table wi differ according to whether the parameters are of an integer type or of a real numb type. Double-length integer type parameters cannot be used (operations will be performe with each parameter being handled as an integer consisting of the lower 16 bits).

Table 4.25 Table of Integer Type LLAG Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	T2	Phase lead time constant	Phase lead time constant (ms)	IN
2	W	T1	Phase lag time constant	Phase lag time constant (ms)	IN
3	W	Y	LLAG output	LLAG output (may also be output to the A register)	OUT
4	W	REM	Remainder	Storage of remainder	OUT
5	W	X	Input value storage	Storage of the input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	LLAG reset	"ON" is input when LLAG is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Table 4.26 Table of Real Type LLAG Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	——	(Reserve)	Reserve register	——
2	F	T2	Phase lead time constant	Phase lead time constant (s)	IN
4	F	T1	Phase lag time constant	Phase lag time constant (s)	IN
6	F	Y	LLAG output	LLAG output (may also be output to the F register)	OUT
8	F	X	Input value storage	Storage of the input value	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	IRST	LLAG reset	"ON" is input when LLAG is reset.	IN
1 to 7	——	(Reserve)	Reserve relay for input	IN
8 to F	——	(Reserve)	Reserve relay for output	OUT

Here, the LLAG operation is expressed as follows:

$$\frac{Y}{X} = \frac{1 + T2 \times S}{1 + T1 \times S} \text{ ; ie. } T1 \times (dY/dt) + Y = T2 + (dX/dt) + X$$

The following operation is performed within the LAG instruction with dt=Ts, dY=Y-Y', and dX=X-X':

$$Y = \frac{T1 \times Y' + (T2 + Ts) \times X - T2 \times X' + REM}{T1 + Ts}$$

X : input value
Y : output value
X' : previous input value
Y' : previous output value
Ts : scan time set value

Y=0, REM=0, and X=0 are output when the LLAG reset (RST) is "ON."

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢. Will be stored if the operation does not start with a ⊢.
*2: Will not be stored if the operation starts with a ⊩. Will be stored if the operation does not start with a ⊩.

[Example(s)] Integer type operation

MW00100 to MW00105 are used for the parameter table.

```
├ MW00010    ← Input value
LLAG  MA00100                        ⇒MW00011
```

Head address of parameter table LLAG output value

Real number type operation

MF00200 to MF00208 are used for the parameter table.

```
├ MF00200    ← Input value
LLAG  MA00200                        ⇒MF00022
```

Head address of parameter table LLAG output value

4.11.9 FGN Instruction

[Format] [Head Address of Parameter Table]

FGN [Register address
Register address with subscript]

[Description] The FGN instruction generates a function curve in accordance with the contents of parameter table that is set in advance. Although the inputs to the FGN instructio can be integer type, double-length integer type, or real number type values, th configuration of the parameter table will differ according to the type of values.

Table 4.27 Table of Integer Type FGN Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	N	Number of data	Number of pairs of X and Y	IN
1	W	X1	Data 1		IN
2	W	Y1	Data 1		IN
3	W	X2	Data 2		IN
4	W	Y2	Data 2		IN
⋮	⋮	⋮	⋮	⋮	⋮
2N-1	W	XN	Data N		IN
2N	W	YN	Data N		IN

Table 4.28 Table of Double-length Integer or Real Type FGN Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	N	Number of data	Number of pairs of X and Y	IN
1	W	—	(Reserve)	Reserve register	IN
2	L/F	X1	Data 1		IN
4	L/F	Y1	Data 1		IN
6	L/F	X2	Data 2		IN
8	L/F	Y2	Data 2		IN
⋮	⋮	⋮	⋮	⋮	⋮
4N-2	L/F	XN	Data N		IN
4N	L/F	YN	Data N		IN

If the data set in the parameter table for the FGN instruction are X_n and Y_n, the data must be set so that $X_n \leqq X_{n+1}$. The FGN instruction searches for an X_n/Y_n pair withir the parameter table for which $X_n \leqq X \leqq X_{n+1}$ and computes the output value Y accordin to the following formula:

$$Y = Y_n + \frac{Y_{n+1} - Y_n}{X_{n+1} - X_n} \times (X - X_n) \quad (1 \leqq {}_n \leqq N-1)$$

The relationship between the data set in parameter table and the input value X anc output value Y will be as shown in Fig. 4.10.

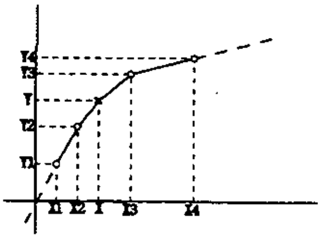


Fig. 4.10 Relationship between Input and Output Values

If an X_n/Y_n pair, which satisfies $X_n \leqq X \leqq X_{n+1}$ for an input value X, does not exist in the parameter table, the result will be as follows:

① If $X < X_1$: $Y=Y_1+\frac{Y_2-Y_1}{X_2-X_1}(X-X_1)$

② If $X > X_1$: $Y=Y_{n+1}+\frac{Y_n-Y_{n-1}}{X_n-X_{n-1}}(X-X_1)$

NOTE

An operation error may occur if the parameters are not set correctly.

A division error will occur if the number of data (number of X/Y pairs) is 0.

When using the FGN instruction for a double-length integer type operation, be sure to execute "⊢ double-length integer type register" immediately before the FGN instruction.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored

* : indeterminate

(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢. Will be stored if the operation does not start with a ⊢.

*2: Will not be stored if the operation starts with a ⊩. Will be stored if the operation does not start with a ⊩.

[Example(s)] Integer type operation (number of data: N=20)

#W00000 to #W00040 are used for the parameter table.

```
⊢ MW00010  ← Input value
FGN  #A00000                          ⇒MW00011
```

Head address of parameter table

Output value

Double-length integer type operation (number of data: N=20)

#L00000 to #L00080 are used for the parameter table.

```
⊢ ML00100  ← Input value
FGN  #A00000                          ⇒ML00102
```

Head address of parameter table

Output value

Real number type operation (number of data: N=20)

#F00000 to #F00080 are used for the parameter table.

```
⊩MF00020  ← Input value
FGN  #A00000                          ⇒MF00022
```

Head address of parameter table

Output value

NOTE

The following form of usage is not allowed.

```
⊢ ML00000  +  10                      ⇒ML00002
   FGN    MA00100                     ⇒ML00004
```

```
⊢ ML00000
   "Comment"
   FGN    MA00100                     ⇒ML00006
```

4.11.10 IFGN Instruction

[Format] [Head Address of Parameter Table]

IFGN [Register address
Register address with subscript]

[Description] The IFGN instruction generates a function curve in accordance with the contents of a parameter table that is set in advance. Although the inputs to the IFGN instructio can be integer type, double-length integer type, or real number type values, the configuration of the parameter table will differ according to the type of values. The parameter tables are the same as those for the FGN instruction.
Refer to the table 4.27 and the table 4.28.

If the data set in the parameter table for the IFGN instruction are X_n and Y_n, the data must be set so that $Y_n \leqq Y_{n+1}$. The IFGN instruction searches for an X_n/Y_n pair within the parameter table for which $Y_n \leqq Y \leqq Y_{n+1}$ for an input value Y and computes the output value X according to the following formula:

$$X = X_n + \frac{X_{n+1} - X_n}{Y_{n+1} - Y_n}\ (Y - Y_n)$$

The relationship between the data set in parameter data and the input value Y and output value X will be as shown in Fig. 4.11.

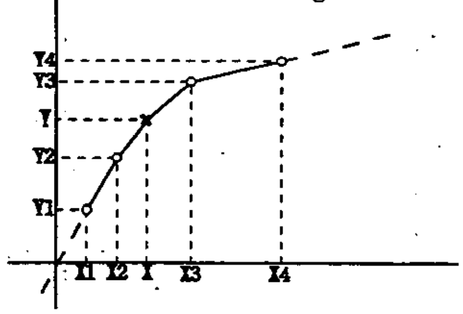


Fig. 4.11 Relationship between Input and Output Values

If an X_n/Y_n pair, which satisfies $Y_n \leqq Y \leqq Y_{n+1}$ for an input value Y, does not exist in the parameter table, the result will be as follows:

① If $Y < Y_1$: $$X = X_1 + \frac{X_2 - X_1}{Y_2 - Y_1}\ (Y - Y_1)$$

② If $Y > Y_1$: $$X = X_1 + \frac{X_n - X_{n-1}}{Y_n - Y_{n-1}}\ (Y - Y_{n-1})$$

NOTE
An operation error may occur if the parameters are not set correctly.
A division error will occur if the number of data (number of X/Y pairs) is 0.
When using the IFGN instruction for a double-length integer type operation, be sure to execute " ⊢ double-length integer type register" immediately before the IFGN instruction.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will not be stored if the operation starts with a ⊢ . Will be stored if the operation does not start with a ⊢ .
*2: Will not be stored if the operation starts with a ⊩ . Will be stored if the operation does not start with a ⊩ .

[Example(s)] Integer type operation (number of data: N=20)
#W00000 to #W00040 are used for the parameter table.

```
├ MW00010   ←Input value
IFGN  #A00000                           ⇒MW00011
       ↑                                     ↑
   Head address of parameter table      Output value
```

Double-length integer type operation (number of data: N=20)
#L00000 to #L00080 are used for the parameter table.

```
├ ML00100   ←Input value
IFGN  #A00000                           ⇒ML00102
       ↑                                     ↑
   Head address of parameter table      Output value
```

Real number type operation (number of data: N=20)
#F00000 to #F00080 are used for the parameter table.

```
├ MF00200   ←Input value
IFGN  #A00000                           ⇒MF00022
       ↑                                     ↑
   Head address of parameter table      Output value
```

NOTE

The following form of usage is not allowed.

```
├ ML00000  +  10                         ⇒ML00002
   IFGN    MA00100                       ⇒ML00004
```

```
├ ML00000
   "Comment"
   IFGN    MA00100                       ⇒ML00006
```

4.11.11 LAU Instruction

[Format] [Head Address of Parameter Table]

LAU [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The LAU instruction is used to perform acceleration and deceleration at a fixe acceleration/deceleration rate upon input of a speed reference (value of the A register The operation is carried out in accordance with the contents of a parameter table tha is set in advance. The input (X) to the LAU operation must be an integer type or rea number type value. The configuration of the parameter table will differ according t whether the parameters are of an integer type or of a real number type. Double length integer type parameters cannot be used (operations will be performed wit each parameter being handled as an integer consisting of the lower 16 bits).

Table 4.29 Table of Integer Type LAU Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	LV	100% input level	Scale of the 100% input	IN
2	W	AT	Acceleration time	Time for acceleration from 0% to 100% (0.1s)	IN
3	W	BT	Deceleration time	Time for deceleration from 100% to 0% (0.1s)	IN
4	W	QT	Quick stop time	Time for quick stop from 100% to 0% (0.1s)	IN
5	W	V	Current speed	LAU output (also output to the A register)	OUT
6	W	DVDT	Current acceleration /deceleration speed	Scaled with the normal acceleration rate being set to 5000.	OUT
7	W	——	(Reserve)	Reserve register	
8	W	VIM	Previous speed reference	For storage of the previous value of the speed reference input	OUT
9	W	DVDTK	Remainder	Scaling coefficient of the current acceleration /deceleration speed (DVDT) (-32768 ~ 32767)	OUT
10	L	REM	Remainder	Remainder of the acceleration/deceleration rate	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	RN	Line is running	"ON" is input while the line is running.	IN
1	QS	Quick stop	"OFF" is input upon quick stop. *1	IN
2	DVDTF	DVDT Operation not executed	"ON" is input at non-execution of DVDT operation.	IN
3	DVDTS	DVDT Operation selection	Selection DVDT operation type	IN
4 to 7	——	(Reserve)	Reserve relay for input	IN
8	ARY	In acceleration	"ON" is output during acceleration.	OUT
9	BRY	In deceleration	"ON" is output during deceleration.	OUT
A	LSP	Zero speed	"ON" is output at a speed of 0.	OUT
B	EQU	Coincidence	"ON" is output when input value = output value.	OUT
C to F	——	(Reserve)	Reserve relay for output	OUT

*1: When the quick stop (QS) is "OFF", the quick stop time is used for the acceleration/deceleration time.

Table 4.30 Table of Real Number Type LAU Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	——	(Reserve)	Reserve register	———
2	F	LV	100% input level	Scale of the 100% input value	IN
4	F	AT	Acceleration time	Time for acceleration from 0% to 100% (s)	IN
6	F	BT	Deceleration time	Time for deceleration from 100% to 0% (s)	IN
8	F	QT	Quick stop time	Time for quick stop from 100% to 0% (s)	IN
10	F	V	Current speed	LAU output (also output to the F register)	OUT
12	F	DVDT	Current acceleration /deceleration speed	Current acceleration/deceleration is output.	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	RN	Line is running	"ON" is input while the line is running.	IN
1	QS	Quick stop	"OFF" is input upon quick stop.	IN
2 to 7	——	(Reserve)	Reserve relay for input	IN
8	ARY	In acceleration	"ON" is output during acceleration.	OUT
9	BRY	In deceleration	"ON" is output during deceleration.	OUT
A	LSP	Zero speed	"ON" is output at a speed of 0.	OUT
B	EQU	Coincidence	"ON" is output when input value = output value.	OUT
C to F	——	(Reserve)	Reserve relay for output	OUT

The following operations are performed inside the LAU instruction:

Integer Type LAU Instruction

$$\text{Acceleration rate (ADV)} = \frac{\text{LV} \times \text{Ts (0.1ms)} + \text{REM}}{\text{AT (0.1s)} \times 1000}$$

When $VI > V'$ ($V' \geqq 0$) :
$V = V' + ADV$; In acceleration (ARY) ON
When $VI < V'$ ($V' \leqq 0$) :
$V = V' - ADV$; In acceleration (ARY) ON

$$\text{Deceleration rate (BDV)} = \frac{\text{LV} \times \text{Ts (0.1ms)} + \text{REM}}{\text{BT (0.1s)} \times 1000}$$

When $VI > V'$ ($V'<0$) :
$V = V' + BDV$; In deceleration (BRY) ON
When $VI < V'$ ($V'>0$) :
$V = V' - BDV$; In deceleration (BRY) ON

$$\text{Quick stop rate (QDV)} = \frac{\text{LV} \times \text{Ts (0.1ms)} + \text{REM}}{\text{QT (0.1s)} \times 1000}$$

When QS=ON ($VI>V'$, $V'<0$) :
$V = V' + QDV$; In deceleration (BRY) ON
When QS=ON ($VI<V'$, $V'>0$) :
$V = V' - QDV$; In deceleration (BRY) ON

V': previous speed output value
Ts : scan time set value (ms)
VI: speed reference input

* If the DVDT operation instruction (DVDTF) is ON, a current acceleration/deceleration operation (DVDT) is performed.
* If DVDTF is OFF, DVDT = 0 is output.
  If DVDTF is ON, a current acceleration/deceleration operation (DVDT) is output after one of the following operations has been performed through DVDT operation selection (DVDTS).

  If DVDTS is ON: $\text{DVDT} = \dfrac{V - V'}{\text{ADV}} \times 5000$

  If DVDTS is OFF: $\text{DVDT} = (V \times \text{DVDTK}) - (V' \times \text{DVDTK})$; DVDTK: DVDT coefficient.

  At V = 0, the zero speed (LSP) is ON, at VI=V, coincidence (EQU) turns ON.
* When the "line is running" (RN) is "OFF," V=0, DVDT=0, and REM=0 are output.

Real Number Type LAU Instruction

$$\text{Acceleration rate (ADV)} = \frac{\text{LV} \times \text{Ts (0.1ms)}}{\text{AT(s)} \times 10000}$$

When VI > V' (V'>0)
V = V' + ADV: "In acceleration" (ARY) is ON
When VI < V' (V'<0)
V = V' − ADV: "In acceleration" (ARY) is ON

$$\text{Deceleration rate (BDV)} = \frac{-\text{LV} \times \text{Ts (0.1ms)}}{\text{BT(s)} \times 10000}$$

When VI < V' (V'>0)
V = V' + BDV: "In deceleration" (BRY) is ON
When VI > V' (V'<0)
V = V' − BDV: "In deceleration" (BRY) is ON

$$\text{Quick stop rate (QDV)} = \frac{-\text{LV} \times \text{Ts (0.1ms)}}{\text{QT(s)} \times 10000}$$

When QS=ON (V'> VI ≧ 0)
V = V' + QDV: "In deceleration" (BRY) is ON
When QS=ON (V'< VI ≦ 0)
V = V' − QDV: "In deceleration" (BRY) is ON

V': previous speed output value
VI: speed reference input
Ts: scan time set value (ms)

The current acceleration/deceleration speed (DVDT) is output after the following operation is carr out:

DVDT=V-V'

When the "line is running" (RN) is "OFF," V=0 and DVDT=0 are output.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will be stored if the operation starts with a ├ . Will not be stored if the operation does not start with a ├ .
*2: Will not be stored if the operation starts with a ╟ . Will be stored if the operation does not start with a ╟ .

[Example(s)] Integer type operation
Use MW00100 to MW00106 for the parameter table.

```
 ├ MW00010  <— Input value
 LAU   MA00100                          ⇒MW00011
          ↑                                  ↑
 Head address of parameter table      LAU output value
```

Real number type operation
Use MF00200 to MF00212 for the parameter table.

```
 ├ MF00200  <— Input value
 LAU   MA00200                          ⇒MF00022
          ↑                                  ↑
 Head address of parameter table      LAU output value
```

4.11.12 SLAU Instruction

[Format] [Head Address of Parameter Table]

SLAU [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The SLAU instruction is used to perform acceleration and deceleration at variable acceleration/deceleration rates upon input of a speed reference (value of the A register). The operation is carried out in accordance with the contents of a parameter table that is set in advance. For integer type SLAU instruction, a positive or a negative value for speed reference input can be entered. For real number type SLAU instruction, only a positive value for speed reference input can be entered. Do not use a negative value therefore. Set it so that the linear acceleration and deceleration time (AT/BT) ≧ S-curve acceleration and deceleration time (AAT/BBT). The input (X) to the SLAU operation must be an integer type or real number type value. The configuration of the parameter table will differ according to whether the parameters are of an integer type or of a real number type. Double-length integer type parameters cannot be used (operations will be performed with each parameter being handled as an integer consisting of the lower 16 bits).

Table 4.31 Table of Integer Type SLAU Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output.*1	IN/OUT
1	W	LV	100% input level	Scale of the 100% input	IN
2	W	AT	Acceleration time	Time for acceleration from 0% to 100% (0.1s)	IN
3	W	BT	Deceleration time	Time for deceleration from 100% to 0% (0.1s)	IN
4	W	QT	Quick stop time	Time for quick stop from 100% to 0% (0.1s)	IN
5	W	AAT	S-curve acceleration time	Time spent in the S-curve region of acceleration (0.01-32.00s)	IN
6	W	BBT	S-curve deceleration time	Time spent in the S-curve region of deceleration (0.01-32.00s)	IN
7	W	V	Current speed	SLAU output (also output to the A register)	OUT
8	W	DVDT1	Current acceleration/deceleration speed 1 (DVDT1)	Scaled with the normal acceleration rate being set to 5000.	OUT
9	W	——	(Reserve)	Reserve register	
10	W	ABMD	Speed increase upon holding	Amount of change in speed after hold instruction and until stabilization.	OUT
11	W	REM1	Remainder	Remainder of the acceleration and deceleration rate	OUT
12	W	——	(Reserve)	Reserve register	
13	W	VIM	Previous speed reference	For storage of the previous value of the speed reference.	OUT
14	L	DVDT2	Current acceleration/deceleration speed 2 (DVDT2)	1000 times of the current acceleration/ deceleration speed	OUT
16	L	DVDT3	Current acceleration/deceleration speed 3(DVDT3)	Current acceleration/deceleration speed(=DVDT2/1000)	OUT
18	L	REM2	Remainder	Remainder of the S-curve region acceleration and deceleration rate	OUT
20	W	REM3	Remainder	Remainder of the current speed	OUT
21	W	DVDTK	DVDT1 coefficient	Scaling coefficient of the current acceleration /deceleration speed 1 (DVDT1) (-32768 to 32767)	IN

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	RN	Line is running	"ON" is input while the line is running.	IN
1	QS^{-}	Quick stop	"OFF" is input upon quick stop.	IN
2	DVDTF	DVDT1 operation not executed	"OFF" is input at non-execution of DVDT1 operation	IN
3	DVDTS	DVDT1 operation selection	Selection of DVDT1 operation type	IN
4 to 7	——	(Reserve)	Reserve relay for input	IN
8	ARY	In acceleration	"ON" is output during acceleration.	OUT
9	BRY	In deceleration	"ON" is output during deceleration.	OUT
A	LSP	Zero speed	"ON" is output at a speed of 0.	OUT
B	EQU	Coincidence	"ON" is output when input value = output value.	OUT
C	EQU	(Reserve)	Reserve relay for output	OUT
D	CCF	Work relay	System internal work relay	OUT
E	BBF	Work relay	System internal work relay	OUT
F	AAF	Work relay	System internal work relay	OUT

Table 4.32 Table of Real Number Type SLAU Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	——	(Reserve)	Reserve register	——
2	F	LV	100% input level	Scale of the 100% input value	IN
4	F	AT	Acceleration time	Time for acceleration from 0% to 100% (s)	IN
6	F	BT	Deceleration time	Time for deceleration from 100% to 0% (s)	IN
8	F	QT	Quick stop time	Time for quick stop from 100% to 0% (s)	IN
10	F	AAT	S-curve acceleration time	Time spent in the S-curve region of acceleration (s)	IN
12	F	BBT	S-curve deceleration time	Time spent in the S-curve region of deceleration (s)	IN
14	F	V	Current speed	SLAU output (also output to the F register)	OUT
16	F	DVDT	Current acceleration/deceleration	Current acceleration/deceleration speed is output.	OUT
18	F	ABMD	Speed increase upon holding	Amount of change in speed after hold instruction and until stabilization.	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	RN	Line is running	"ON" is input while the line is running.	IN
1	QS	Quick stop	"OFF" is input upon quick stop.	IN
2 to 7	——	(Reserve)	Reserve relay for input	IN
8	ARY	In acceleration	"ON" is output during acceleration.	OUT
9	BRY	In deceleration	"ON" is output during deceleration.	OUT
A	LSP	Zero speed	"ON" is output at a speed of 0.	OUT
B	EQU	Coincidence	"ON" is output when input value = output value.	OUT
C to F	——	(Reserve)	Reserve relay for output	OUT

The following operations are performed inside the SLAU instruction:

Integer Type SLAU Instruction

$$\text{Acceleration rate (ADV)} = \frac{\text{(LV} \times \text{Ts(0.1ms)+REM1)}}{\text{AT(0.1s)} \times 1000}$$

When VI > V' (V' ≧ 0) outside the S-curve region (ADVS > ADV):
V = V' + ADV; In acceleration (ARY) ON
When VI < V' (V' ≦ 0):
V = V'−ADV; In acceleration (ARY) ON

$$\text{Deceleration rate (BDV)} = \frac{\text{(LV} \times \text{Ts(0.1ms)+REM1)}}{\text{BT(0.1s)} \times 1000}$$

When VI > V' (V'< 0) outside the S-curve region (BDVS < BDV):
V = V' + BDV; In deceleration (BRY) O
When VI < V' (V'> 0):
V = V'−BDV; In deceleration (BRY) ON

$$\text{Quick stoppage rate (QDV)} = \frac{\text{(LV} \times \text{Ts(0.1ms)+REM1)}}{\text{QT(0.1s)} \times 1000}$$

When QS=ON (VI > V', V'<0) :
V = V' + QDV; In deceleration (BRY) ON
When QS=ON (VI < V', V'>0) :
V = V'−QDV; In deceleration (BRY) ON
(Note) At quick stop, the movement is not curve but linear (same as during LA quick stop).

Acceleration rate in the S-curve region (ADVS) = ADVS' ± AADVS

$$\text{AADVS} = \frac{\text{ADV} \times \text{Ts(0.1ms)+REM2}}{\text{AAT(0.01s)} \times 100}$$

When VI > V' (V' ≧ 0) inside the S-curve region (ADVS < ADV):
V = V' + ADVS; In acceleration (ARY) ON
When VI < V' (V' ≦ 0):
V = V'−ADVS; In acceleration (ARY) ON

Deceleration rate in the S-curve region (BDVS) = BDVS' ± BBDVS

$$BBDVS = \frac{BDV \times Ts(0.1ms) + REM2}{BBT(0.01s) \times 100}$$

When VI > V' (V'< 0) inside the S-curve region (BDVS < BDV):
V = V' + BDVS; In deceleration (BRY) ON
When VI < V' (V'>0):
V = V' −BDVS; In deceleration (BRY) ON

V' : previous speed output value
Ts : scan time set value (ms)
VI: speed reference input

- * If the DVDT operation instruction (DVDTF) is ON, a current acceleration/deceleration speed operation 1 (DVDT1) is performed.
- * If DVDTF is OFF, DVDT1 = 0 is output.
  IF DVDTF is ON, a current acceleration/deceleration speed operation 1 (DVDT1) is output after one of the following operations has been performed through DVDT1 operation selection (DVDTS).

If DVDTS is ON: $DVDT1 = \frac{V - V'}{ADV} \times 5000$

If DVDTS is OFF: $(V \times DVDTK) - (V' \times DVDTK)$; DVDTK: DVDT coefficient.

- * The current acceleration/deceleration speed 2 (DVDT2) is output as follows:
  During acceleration inside the S-curve region : DVDT2 = ± ADVS
  During acceleration outside the S-curve region : DVDT2 = ± ADV
  During deceleration inside the S-curve region : DVDT2 = ± BDVS
  During deceleration outside the S-curve region : DVDT2 = ± BDV

- * The speed increase upon holding (ABMD) is output after the following operation is performed.

$$ABMD = \frac{DVDT2' \times DVDT2'}{2 \times AADVS(BBDVS)};$$

DVDT2' = Current acceleration/deceleration speed 2 (DVDT2) previous value

- * At V = 0, the zero speed (LSP) is ON, at VI=V, coincidence (EQU) turns ON.
- * When the line running signal (RN) is "OFF," V=0, DVDT1=0, DVDT2=0, DVDT3=0, ABMD=0, REM1=0, REM2=0, and REM3=0 are output.

Real Number Type SLAU Instruction

$$\text{Acceleration rate (ADV)} = \frac{LV \times Ts\ (0.1ms)}{AT(s) \times 10000}$$

When VI > V' (V'> 0) outside the S-curve region (ADVS > ADV): V = V' + ADV

$$\text{Deceleration rate (BDV)} = \frac{-LV \times Ts\ (0.1ms)}{BT(s) \times 10000}$$

When VI < V' (V'> 0) outside the S-curve region (BDVS < BDV): V = V' + BDV

$$\text{Quick stop rate (QDV)} = \frac{-LV \times Ts\ (0.1ms)}{QT(s) \times 10000}$$

When QS=ON (V'> VI) :
V = V' + QDV

Acceleration rate in the S-curve region (ADVS) = ADVS' ± AADVS:
where ADVS' = ADVS previous value

$$AADVS = \frac{ADV \times Ts\ (0.1ms)}{AAT(s) \times 10000}$$

When VI > V' (V'> 0) inside the S-curve region (ADVS < ADV): V = V' + ADVS

Deceleration rate in the S-curve region (BDVS) = BDVS' ± BBDVS:
where BDVS = BDVS previous value

$$BBDVS = \frac{BDV \times Ts\ (0.1ms)}{BBT(s) \times 10000}$$

When VI < V' (V'> 0) inside the S-curve region (BDVS > BDV): V = V' + BDVS

V' : previous speed output value
VI : speed reference input
Ts : scan time set value (ms)

SLAU Instruction

The current acceleration/deceleration speed (DVDT) is output after the following operation is carr out:

During acceleration inside S-curve region : DVDT = ADVS
During acceleration outside S-curve region : DVDT = ADV
During deceleration inside S-curve region : DVDT = BDVS
During deceleration outside S-curve region : DVDT = BDV

The speed increase upon holding (ABMD) is output after the following operation is performed.

$$\mathrm{ABMD} = \frac{\mathrm{DVDT} \times \mathrm{DVDT}}{2 \times \mathrm{AADVS(BBDVS)}}$$

When the "line is running" signal (RN) is "OFF", V=0, DVDT=0, and ABMD=0 are output.

[Operation of the Register]

A	F	B	I	J
*1	*2	○	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

*1: Will be stored if the operation starts with a ├ . Will not be stored if the operation does not start with a ├ .
*2: Will not be stored if the operation starts with a ╟. Will be stored if the operation does not start with a ╟.

[Example(s)] Integer type operation
Use MW00100 to MW00111 for the parameter table.

```
├ MW00010  ←Input value
SLAU  MA00100                        ⇒MW00011
      ↑                                  ↑
  Head address of parameter table    SLAU output value
```

Real number type operation
MF00200 to MF00218 are used for the parameter table.

```
╟MF00200   ←Input value
SLAU  MA00200                        ⇒MF00022
      ↑                                  ↑
  Head address of parameter table    SLAU output value
```

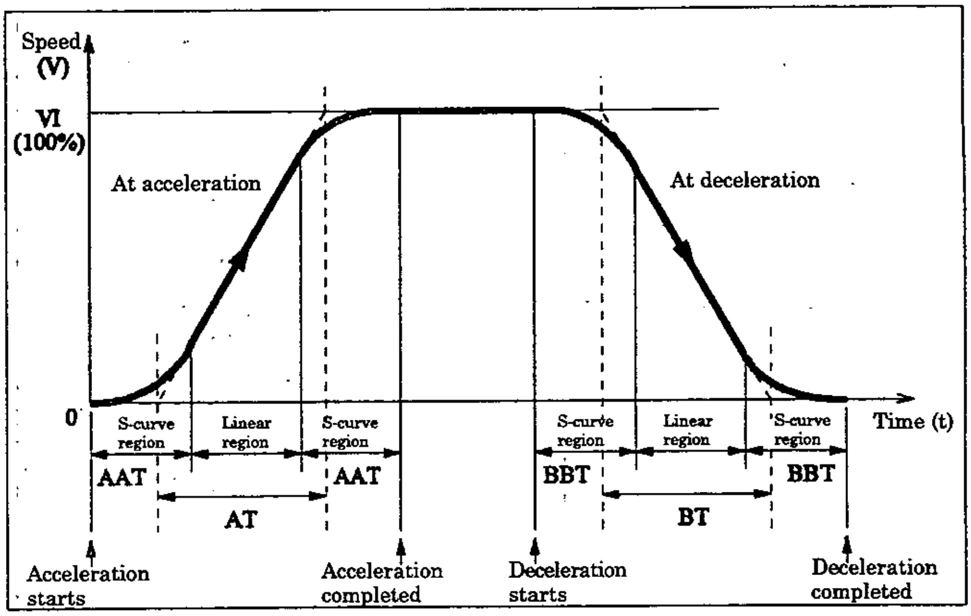


Fig. 4.12 Motion by SLAU

.11.13 PWM Instruction

[Format] [Head Address of Parameter Table]

PWM [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)]

[Description] The PWM instruction converts the value of the A register to PWM as input value (-100.00 to 100.00%, units: 0.01%), and the result is output to the B register and the parameter table.
Double-length type integer operations and real number type operations are not allowed.
Time of ON output and number of ON outputs are expressed as follows.

$$\text{Time of ON output} = \frac{\text{PWMT}(X+10000)}{20000}$$

$$\text{Number of ON outputs} = \frac{\text{PWMT}(X+10000)}{\text{Ts}\times 20000}$$

X: input value

Ts: scan time set value (ms)

[When 100.00% is input: all ON
When 0% is input: 50% duty (50% ON)
When -100.00% is input: all OFF]

When the PWM reset (PWMRST) is "ON", all internal operations are reset. PWM operations are performed with that instant as the starting point. After powering up, first turn "ON" PWMRST and clear internal operations. Then use the PWM instruction.

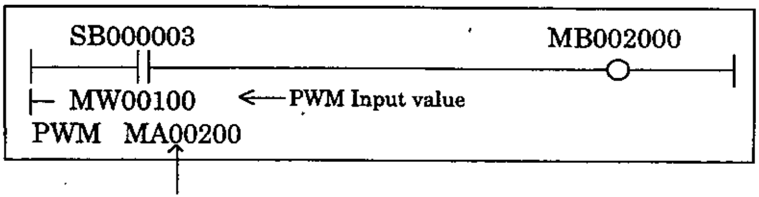
Table 4.33 Table of PWM Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	W	RLY	Relay I/O	Relay input, relay output *1	IN/OUT
1	W	PWMT	PWM cycle	PWM cycle (1 ms) (1 to 32767 ms)	IN
2	W	ONCNT	ON output setting timer	ON output setting timer (1 ms)	OUT
3	W	CVON	ON output count timer	ON output count timer (1 ms)	OUT
4	W	CVONREM	ON output count timer remainder	ON output count timer remainder (0.1 ms)	OUT
5	W	OFFCNT	OFF output setting timer	OFF output setting timer (1 ms)	OUT
6	W	CVOFF	OFF output count timer	OFF output count timer (1 ms)	OUT
7	W	CVOFFREM	OFF output count timer remainder	OFF output count timer remainder (0.1 ms)	OUT

*1: Relay I/O Bit Assignment

BIT	Symbol	Name	Specification	I/O
0	PWMRST	PWM reset	"ON" is input when PWM is reset	IN
2 to 7	——	(Reserve)	Reserve relay for input	IN
8	PWMOUT	PWM output	PWM is output (two-value output: ON=1, OFF=0)	OUT
9 to F	——	(Reserve)	Reserve relay for output	OUT

[Example(s)] MW00100 is used as PWM input and MW00200 to MW00207 as a parameter table.



Head address of parameter table

PWM reset with the first scan of DWG.L (SB000001 when used with DWG.H)

4.12 Table Data Operation Instructions

When an error occurs at the execution of table data operation instruction, an error code is set to A register and B register is turned ON. For the error codes, refer to Table 4.34.

Table 4.34 List of Errors

Error code	Error name	Contents
0001H	Reference table not defined	The target table has not been defined.
0002H	Outside row number range	The row numbers of the table element are not i the range of the target table.
0003H	Outside column number range	The column numbers of the table element are nc in the range of the target table.
0004H	Wring number of elements	The number of target elements is not correct
0005H	Insufficient space in storage destination	Area for storing is not adequate.
0006H	Wrong element format	The format of the specified element is wrong.
0007H	Cue buffer error	An attempt is made to read the cue buffer when is empty, or the buffer is written to by pointe advance when it is full.
0008H	Cue table error	The designated table is not a cue type table.
0009H	System error	An unexpected error is detected internally in th system during instruction execution.

4.12.1 Block Read Instruction (TBLBR)

[Format]

TBLBR [Transfer source table name] , [Head Address of Transfer Destination Data] [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)] , [Head Address of Parameter Tab] [Register address
Register address with subscript]

[Description] The block read instruction consecutively reads, in block format, elements of the file register table specified by table name, row number, and column number. The instruction then stores the elements in a consecutive region beginning with the specified register. The type of the elements read is automatically judged based on the table specified. The format of the register stored at is ignored. The read value is stored according to the table element format without format conversion.

In referencing a table, if there is anything invalid in the name, row number, column number, or insufficient data length storage, an error is reported, and the data is not read. The contents of the register for storage are kept.

Upon normal completion, the number of words transmitted is set in the A register, the B register turns OFF. When an error occurs, an error code is set in A register, and B register turns ON. For error codes, refer to Table 4.34.

Table 4.35 Table of Block Read Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW1	Table element beginning row number	Target table element beginning row number (1 to 65535)	IN
2	L	COL1	Table element beginning column number	Target table element beginning column number (1 to 32767)	IN
4	W	RLEN	Number of row elements	Number of row elements (1 to 32767)	IN
5	W	CLEN	Number of column elements	Number of column elements (1 to 32767)	IN

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored ×: not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] From the table defined as TABLE 1, using DW00010 to DW00013 as a parameter table, data (element type is integer type) from the starting table element position to the end position are stored in block form in the area starting from MW00100.

```
                                         MB000000
TBLBR TABLE1, MA00100, DA00010     ────────○──┤
                                       ⇒ MW00011
```

12.2 Block Write Instruction (TBLBW)

[Format]

		[Head Address of Transfer Destination Data]		[Head Address of Parameter Table]
TBLBW	[Transfer source table name] ,	[Register address (except for # and C registers) Register address with subscript (except for # and C registers)]	,	[Register address Register address with subscript]

[Description] The block write instruction consecutively stores a consecutive region beginning with the designated register, using block format in elements of the file register table specified by table name, row number, and column number. The data is processed assuming the form of the elements in the storage and the format of the storage source register conform.

In referencing a table, if there is anything invalid in the name, row number, column number, or insufficient length at data destination, an error is reported, and the data is not read. The contents of the register for storage are kept.

Upon normal completion, the number of words transmitted is set in the A register, the B register turns OFF. When an error occurs, an error code is set in A register, and B register turns ON. For error codes, refer to Table 4.34.

Table 4.36 Table of Block Write Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW1	Table element beginning row number	Target table element beginning row number (1 to 65535)	IN
2	L	COL1	Table element beginning column number	Target table element beginning column number (1 to 32767)	IN
4	W	RLEN	Number of row elements	Number of row elements (1 to 32767)	IN
5	W	CLEN	Number of column elements	Number of column elements (1 to 32767)	IN

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] From the table defined as TABLE 1, with DW00010 to DW00013 as a parameter table, data (element type is integer type) from the starting table element position to the end position are stored in block form in the area beginning with MW00100.

```
                                      MB000000
TBLBW  TABLE1, MA00100, DA00010    ──────○──
                                     ⇒ MW00011
```

4.12.3 Row Search Instruction: Vertical Direction (TBLSRL)

[Format]

TBLSRL [Name of table to be searched.] , [Head Address of Search Data] [Register address (except for # and C registers)
Register address with subscript (except for # and C registers)] , [Head Address of Parameter Table] [Register address
Register address with subscript]

[Description] The row search instruction searches the column element of a file register table specifie by table name, row number, and column number, and if there is data which matche the data of the register, reports that row number. The type of the data to be searche is automatically judged based on the table specified.
In referencing a table, if there is anything invalid in the name, row number, colum number, or insufficient length at data destination, an error is reported.
Upon normal completion, the B register turns OFF. If matching column element were found, a "1" is set in the search result, and in register A, the corresponding ro number is set. If matching column elements were not found, a "0" is set in the searc result. When an error occurs, an error code is set in A register, and B register turn ON. For error codes, refer to Table 4.34.

Table 4.37 Table of Row Search Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/
0	L	ROW1	Head row number of table element	Head row number of the target table element (1 to 65535)	IN
2	L	ROW2	Last row number of table element	Last row number of the target table element (1 to 65535)	IN
4	L	COLUMN	Table element column number	Column number of the target table element (1 to 32767)	IN
6	W	FIND	Search result	Search results 0: No matching row 1: Matching row exists	OU

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The table defined as TABLE1 is searched for data which matches MW00100 (wher the type of the searched table is integer) with DW00010 to DW00013 as a paramete table.

```
                                         MB000000
TBLSRL  TABLE1, MA00100, DA00010  ────────○──┤
                                        ⇒ MW00011
```

.12.4 Column Search Instruction: Horizontal Direction (TBLSRC)

[Format]

	[Head Address of Search Data]	[Head Address of Parameter Table]
TBLSRC [Name of table to be searched.],	[Register address (except for # and C registers) Register address with subscript (except for # and C registers)],	[Register address Register address with subscript]

[Description] The column search instruction searches the row element of a file register table specified by table name, row number, and column number, and if there is data which matches the data of the register, reports that column number. The type of the data to be searched is automatically judged based on the table specified.
In referencing a table, if there is anything invalid in the name, row number, column number, or insufficient length at data destination, an error is reported.
Upon normal completion, the B register turns OFF. If matching row elements were found, a "1" is set in the search result, and in register A, the corresponding column number. If matching column elements were not found, a "0" is set in the search result. When an error occurs, an error code is set in A register, and B register turns ON. For error codes, refer to Table 4.34.

Table 4.38 Table of Column Search Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW	Table element row number	Row number of the target table element (1 to 65535)	IN
2	L	COLUMN1	Head column number of table element	Head column number of the target table element (1 to 32767)	IN
4	L	COLUMN2	Last column number of table element	Last column number of the target table element (1 to 32767)	IN
6	W	FIND	Search result	Search results 0: No corresponded column 1: Corresponded column exists	OUT

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The table defined as TABLE1 is searched for data which matches MW00100 (when the type of the searched table is integer) with DW00010 to DW00013 as a parameter table.

```
                                           MB000000
TBLSRC  TABLE1, MA00100, DA00010  ————————○——
                                         ⇒ MW00011
```

4.12.5 Block Clear Instruction (TBLCL)

[Format]

[Head Address of
Parameter Table]

TBLCL [Target table name], [Register address
Register address with
subscript]

[Description] The block clear instruction clears the data of the block element of a file register table specified by table name, row number, and column number. If the type of the element is a character string, a space is written, and a 0 is written if it is a numerical value.If both the head row number and the head column number of the table element destination are 0, the entire table will be cleared. In referencing a table, if there is anything invalid in the name, row number, column number, or insufficient length at data destination, an error is reported, and the data is not read.
Upon normal completion, the number of words cleared is set in the A register, the B register turns OFF. When an error occurs, an error code is set in A register, and B register turns ON. For error codes, refer to Table 4.34.

Table 4.39 Table of Block Clear Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW	Head row number of table element	Head row number of the target table element (0 to 65535)	IN
2	L	COLUMN	Head column number of table element	Head column number of the target table element (1 to 32767)	IN
4	W	RLEN	Number of row elements	Number of row elements (1 to 32767)	IN
5	W	CLEN	Number of column elements	Number of column elements (1 to 32767)	IN

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The designated block in the table defined as TABLE1 is cleared using DW00010 to DW00013 as a parameter table.

```
                                      MB000000
TBLCL  TABLE1, DA00010             ────────○─
                                    ⇒ MW00011
```

12.6 Inter Table Block Transfer Instruction (TBLMV)

[Format]

TBLMV [Transfer source table name], [Transfer destination table name], [Head Address of Parameter Table]
[Register address
Register address with subscript]

[Description] The inter table block transfer instruction transfers the data of a block element of a file register table specified by table name, row number, and column number to another block. Transfers both between different tables and transfers within the same table are possible, but if the type of the transfer source and transfer destination are not identical, an error is reported, and the data cannot be written.
In referencing a table, if there is anything invalid in the name, row number, column number, or insufficient length at data destination, an error is reported, and the data is not read.
Upon normal completion, the number of words transferred is set in the A register, the B register turns OFF. When an error occurs, an error code is set in A register, and B register turns ON. For error codes, refer to Table 4.34.

Table 4.40 Table of Inter Table Block Transfer Instruction Parameters

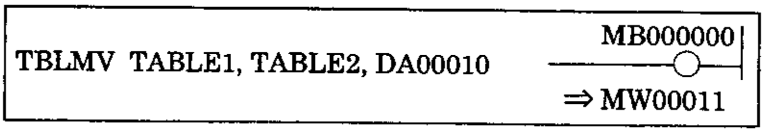
ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW1	Head row number of table element	Head row number of the transfer source table element (1 to 65535)	IN
2	L	COLUMN1	Head column number of table element	Head column number of the transfer source table element (1 to 32767)	IN
4	W	RLEN	Number of row elements	Number of transfer row elements (1 to 32767)	IN
5	W	CLEN	Number of column elements	Number of transfer column elements (1 to 32767)	IN
6	L	ROW2	Head row number of table element	Head row number of the transfer destination table element (1 to 65535)	IN
8	L	COLUMN2	Head column number of table element	Head column number of the transfer destination table element (1 to 32767)	IN

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] There are tables defined as TABLE1 and TABLE2. The designated block in TABLE1 is transferred to the designated block in TABLE2 using DW00010 to DW00015 as a parameter table.



TBLMV TABLE1, TABLE2, DA00010

4.12.7 Cue Table Read Instruction (QTBLR, QTBLRI)

[Format]

		[Head Address of Transfer Destination Data]	[Head Address of Parameter Table]
QTBLR QTBLRI	Transfer source table name ,	Register address (except for # and C registers) Register address with subscript (except for # and C registers) ,	Register address Register address with subscript

[Description] The cue table read instruction continuously reads column elements of a file registe table specified by table name, row number, and column number, and stores it i consecutive areas beginning with the specified register. The type of the element to b read is automatically judged based on the table specified. The type of the register fo storage is ignored. The read value is stored according to the table element forma without type conversion. The cue table read pointer is not changed by a QTBLI instruction. The cue pointer is advanced one row by a QTBLRI instruction. I referencing a table, if there is anything invalid in the name, row number, colum number, insufficient length at data destination, or the cue buffer is empty, an error i reported, the data is not read, and the cue pointer does not advance. The contents o the register for storage are kept.

Upon normal completion, the number of words transferred is set in the A register, th B register turns OFF. When an error occurs, an error code is set in A register, and I register turns ON. The pointer value does not change. For error codes, refer to Tabl 4.34.

Table 4.41 Table of Cue Table Read Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/(
0	L	ROW	Relative row numbers for table elements	Relative column number of the target table element (0 to 65535)	IN
2	L	COLUMN	Head column number of table element	Head column number of the target table element (1 to 32767)	IN
4	W	CLEN	Number of column elements	Number of column elements to be continuously read out (1 to 32767)	IN
5	W	Reserve			
6	L	RPTR	Read pointer	Read pointer of the cue after execution	OU
8	L	WPTR	Write pointer	Write pointer of the cue after execution	OU

By setting relative row numbers for the table elements, the actual row position read will vary as in Table 4.42.

Table 4.42 Settings for Relative Row Numbers for Table Elements

Relative row numbers	Row read	Remark
0	Read pointer row	Pointer advance for QTBLRI only
1	Write pointer row	No pointer advance
2	(Write pointer row)-1	No pointer advance
3	(Write pointer row)-2	No pointer advance
⋮	⋮	⋮
n	(Write pointer row)-(n-1)	No pointer advance

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Column element data (element format assumed to be integer) from the table defined as TABLE1 is stored for the number of column elements beginning with MW0010 using DW00010 to DW00012 as a parameter table.

```
                                      MB000000
QTBLRI TABLE1, MA00100, DA00010   ------O------|
                                  ⇒ MW00011
```

12.8 Cue Table Write Instruction (QTBLW, QTBLWI)

[Format]

		[Head Address of Transfer Source Data]	[Head Address of Parameter Table]
QTBLW QTBLWI	Transfer destination table name ,	Register address Register address with subscript ,	Register address Register address with subscript

[Description] The cue table write instruction continuously reads data from consecutive areas beginning with the specified register, and writes it to column elements of a file register table specified by table name, row number, and column number. Data is processed assuming the format of the element of the table at the location to be stored at is the same as the type of the register storage source.
The cue table write pointer is not changed by a QTBLW instruction. The cue pointer is advanced one row by a QTBLWI instruction.
In referencing a table, if there is anything invalid in the name, row number, column number, insufficient length at data destination, or the cue buffer is full, an error is reported, the data is not written, and the cue pointer does not advance.
Upon normal completion, the number of words transferred is set in the A register, the B register turns OFF. When an error occurs, an error code is set in A register, and B register turns ON. The pointer value does not change. For error codes, refer to Table 4.34.

Table 4.43 Table of Cue Table Write Instruction Parameters

ADR	Type	Symbol	Name	Specification	I/O
0	L	ROW	Relative row numbers for table elements	Relative column number of the target table element (0-65535)	IN
2	L	COLUMN	Head column number of table element	Head column number of the target table element (1 to 32767)	IN
4	W	CLEN	Number of column elements	Number of column elements to be continuously written (1 to 32767)	IN
5	W	Reserve			
6	L	RPTR	Read pointer	Read pointer of the cue after execution	OUT
8	L	WPTR	Write pointer	Write pointer of the cue after execution	OUT

By setting relative row numbers for the table elements, the actual row position write will vary as in Table 4.44.

Table 4.44 Settings for Relative Row Numbers for Table Elements

Relative row numbers	Row write	Remark
0	Write pointer row	Pointer advance for QTBLWI only
1	Write pointer row	No pointer advance
2	(Write pointer row)-1	No pointer advance
3	(Write pointer row)-2	No pointer advance
⋮	⋮	⋮
n	(Write pointer row)-(n-1)	No pointer advance

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] Integer form consecutive data for the number of column elements beginning with MW00100 is written in column element data in the table defined as TABLE1 using DW00010 to DW00013 as a parameter table.

```
                                        MB000000
QTBLWI TABLE1, MA00100, DA00010     ---------( )--
                                    ⇒ MW00011
```

4.12.9 Cue Pointer Clear Instruction (QTBLCL)

[Format] QTBLCL [Transfer source table name]

[Description] The cue pointer clear instruction returns the cue read and cue write pointer of the fil register table specified by table name to initial status (first row).
Upon normal completion, a "0" is set in the A register, the B register turns OFF
When an error occurs, an error code is set in A register, and B register turns ON.
For error codes, refer to Table 4.34.

[Operation of the Register]

A	F	B	I	J
×	○	×	○	○

○: stored × : not stored
* : indeterminate
(Stored or not stored depending on the case.)

[Example(s)] The cue read and cue write pointer of TABLE1 are reset to initial status.

```
                                   MB000000
 QTBLCL TABLE1              ─────────○──┤
                                  ⇒ MW00011
```

5 SFC PROGRAMMING

The programming method, in which SFC's (sequential function charts) are used, is described in this chapter.

5.1 Configuration of an SFC Program

As shown in Fig. 5.1, an SFC program is composed of an SFC flowchart, an SFC action box, and an SFC output definition time chart.

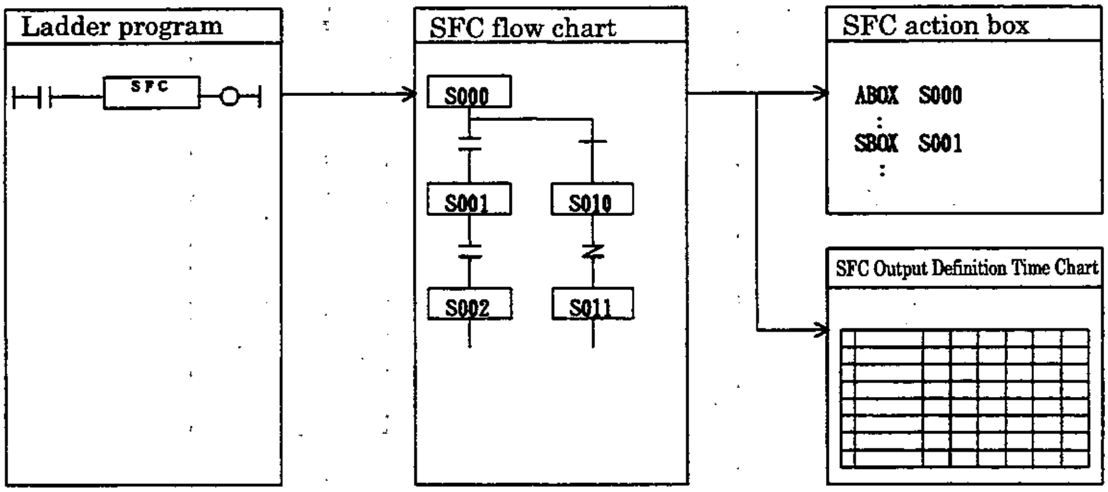


Fig. 5.1 Configuration of an SFC Program

5.2 Execution of SFC

As shown in Fig. 5.2, the SFC program is executed by the SFC instruction in the ladder program. The SFC program is executed through step transition control, which is managed by the use of system step numbers. The system automatically assigns a system step number to each step name. The assigned system step number can be checked at the SFC Output Definition Time Chart screen of the CP-717. Since the system step number will be changed when an SFC step is added or deleted, do not make changes in the SFC flowchart while the line is running.

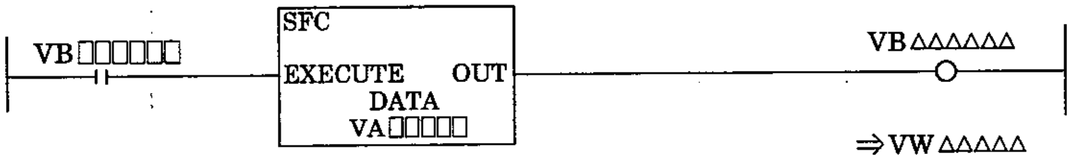


Fig. 5.2 SFC Instruction

Table 5.1 I/O Registers

I/O Symbol	Registers that can be designated (V=)	Description
VB□□□□□□ (EXECUTE)	S, I, O, M, D, C, #	· SFC execution instruction · Execution control (step transition control) of the SFC is carried out when this register is ON. · The current system step will always be set to the initial step when this register is set to OFF.
VA□□□□□	M, D	· Designation of the head register number of the register area for the SFC system operation. · See Section 5.3, "SFC System Operation Register" for details.
VB△△△△△△ (OUT)	O, M, D	· SFC step transition output (becomes ON when step transition is carried out). · Within a parallel process, this will contain the result of the final parallel process sequence.
VW△△△△△	O, M, D	· Designation of the user step number output corresponding to the current system step · See Section 5.7, "Step Name Designation Method" for details.

.3 SFC System Operation Registers

The system operation registers necessary for the execution of an SFC program are set up as shown in Table 5.2. When an SFC program is to be used, these registers may not be used for other purposes.

Table 5.2 Assignment of the SFC System Process Registers

Register No.	Name	Description
VW□□□ 00	System step - current value	System step number when an ordinary process is being carried out[*1].
01	System step - previous value	System step number prior to transition when an ordinary process is being carried out[*1].
02	Transition timer for count	Count register used for the transition timer when an ordinary process is being carried out[*1].
03	User step search input	For searching for the system step corresponding to the user step. User step no. : Bit 0 to Bit E. search execution command : Bit F.
04	SFC output bit [*3] - 1	Output data from the SFC Output Definition Time Chart (0 to 15).
05	SFC output bit [*3] - 2	Output data from the SFC Output Definition Time Chart (16 to 31).
06	SFC output bit [*3] - 3	Output data from the SFC Output Definition Time Chart (32 to 47).
07	SFC output bit [*3] - 4	Output data from the SFC Output Definition Time Chart (48 to 63).
08 09	For SFC parallel process control	For system use
10 : 17	For SFC function operation	Step number of each process when a parallel process is being carried out.[*2]
18 : 25	For SFC function operation	Count register used for the transition timer for each parallel process when a parallel process is being carried out.[*2]
26	SFC output bit [*3] - 5	Output data from the SFC Output Definition Time Chart (64 to 79).
27	SFC output bit [*3] - 6	Output data from the SFC Output Definition Time Chart (80 to 95).
28	SFC output bit [*3] - 7	Output data from the SFC Output Definition Time Chart (96 to 111).
29	SFC output bit [*3] - 8	Output data from the SFC Output Definition Time Chart (112 to 127).

[*1] : Ordinary process : Only a single step is processed.

[*2] : Parallel process : A plurality of steps are processed simultaneously and in parallel by parallel process branching.

[*3] : SFC output bit : In parallel processing, the logical sum (OR) of the outputs of the parallel process steps is output.

5.4 SFC Flowchart

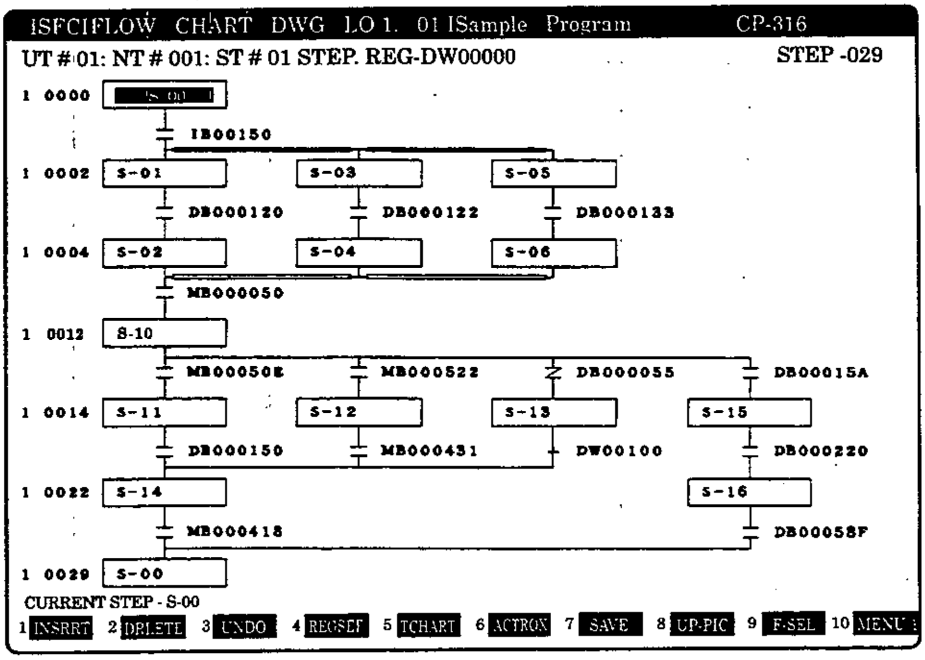
The SFC flowchart is prepared using steps, transition conditions, and connection designations. Th sequence proceeds from the initial step in accordance with the transition conditions and the transitio to the next step is performed when conditions are satisfied. The transition of the execution of th steps is performed from top to bottom. If the SFC program cannot be prepared with just one flo chart, it can be divided into a plurality of flowcharts (or composed of subroutines).

■ **Step** : One step in a sequence.
- Expressed with a box (□) and a step name (with 6 or less alphanumeric or symboli characters).
- A step can be in the logic state of ON (active) or OFF (inactive) and when a step becomes O (active), the SFC Action Box associated with the step is executed.
- A system step number controlled by the system is assigned to the step automatically. The SF is controlled by means of these step numbers.

■ **Transition condition** : The logic condition that must be satisfied for step transition.
- NO contact condition (╪) : Step transition is carried out when ON.
- NC contact condition (⧣) : Step transition is carried out when OFF.
- Timer transition condition (+) : Step transition is carried out after the set time.

■ **Single-token Structure (designation of ordinary branching connection)**
- An ordinary process branching or convergence is expressed with a single lin (—) and only one of the branch processes is executed. If a plurality of conditions are satisfie the condition at the left side has priority.
- Branching designation, convergence designation, and converging connection designation ma be used.

■ **Multi-token Structure (designation of parallel branching connection)**
- A parallel process branching or convergence is expressed with a double line (═) and paralle processes are executed simultaneously and in parallel.
- Branching designation, convergence designation, and converging connection designation ma be used.
- The number of parallel process branches must 6 or less.
- At the branching point of a parallel process, the parallel processes are started simultaneousl after the transition to the step.
- At the parallel process convergence point, the transition to the step following the convergenc point is carried out when all of the parallel processes have reached the step prior to th convergence point and the transition conditions are satisfied.



ISFCIFLOW CHART DWG LO 1. 01 ISample Program CP-316

UT # 01: NT # 001: ST # 01 STEP. REG-DW00000 STEP -029

CURRENT STEP - S-00

1 INSRRT 2 DELETE 3 UNDO 4 REGSET 5 TCHART 6 ACTBOX 7 SAVE 8 UP-PIC 9 F-SEL 10 MENU

.5 SFC Action Box

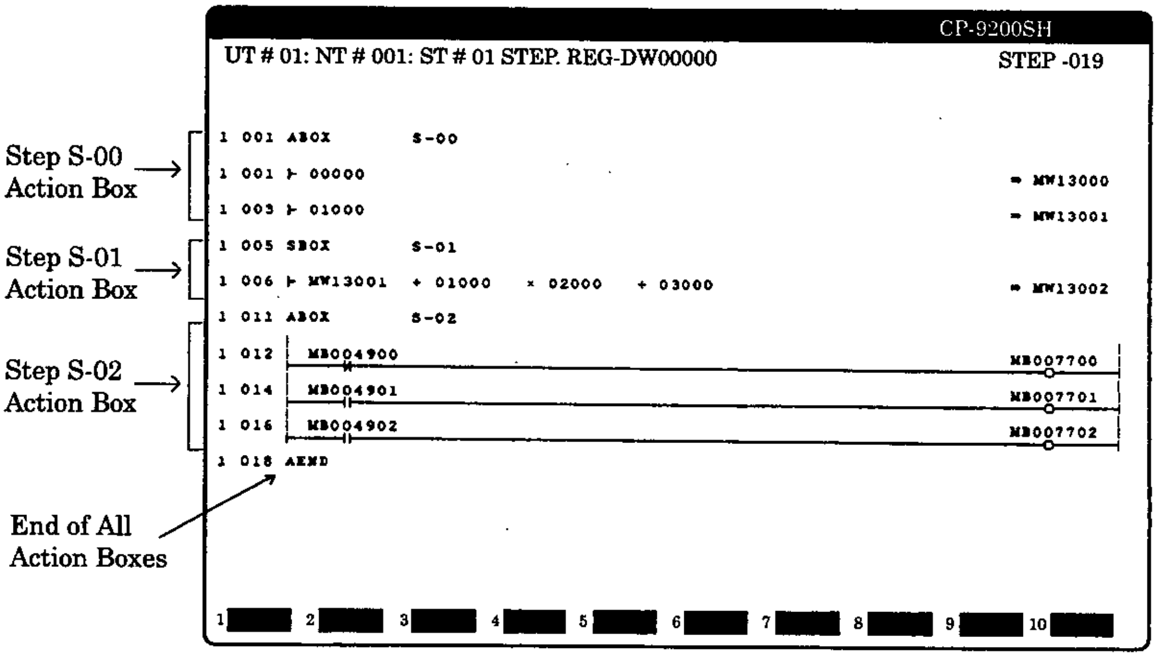
The SFC Action Box is prepared using the ABOX and SBOX instructions. The program, that is to be executed when a step in the SFC flowchart becomes ON (active), is prepared in the SFC Action Box. This program is prepared with ladder programming language and text type language. One step of an SFC Action Box Program will consist of the instruction sequence up to the ABOX instruction or SBOX instruction of the next step and the SFC Action Box Program comprising all steps is ended with an AEND instruction .
It is not necessary to create an action box for each step. An Action Box is created only for steps which require processing.

■ ABOX Instruction

With this instruction , the corresponding program is executed on each scan from the point at which the corresponding step is entered and until the transition to the next step is carried out.

■ SBOX Instruction

With this instruction , the corresponding program is executed just once at the point of the transition to the corresponding step.

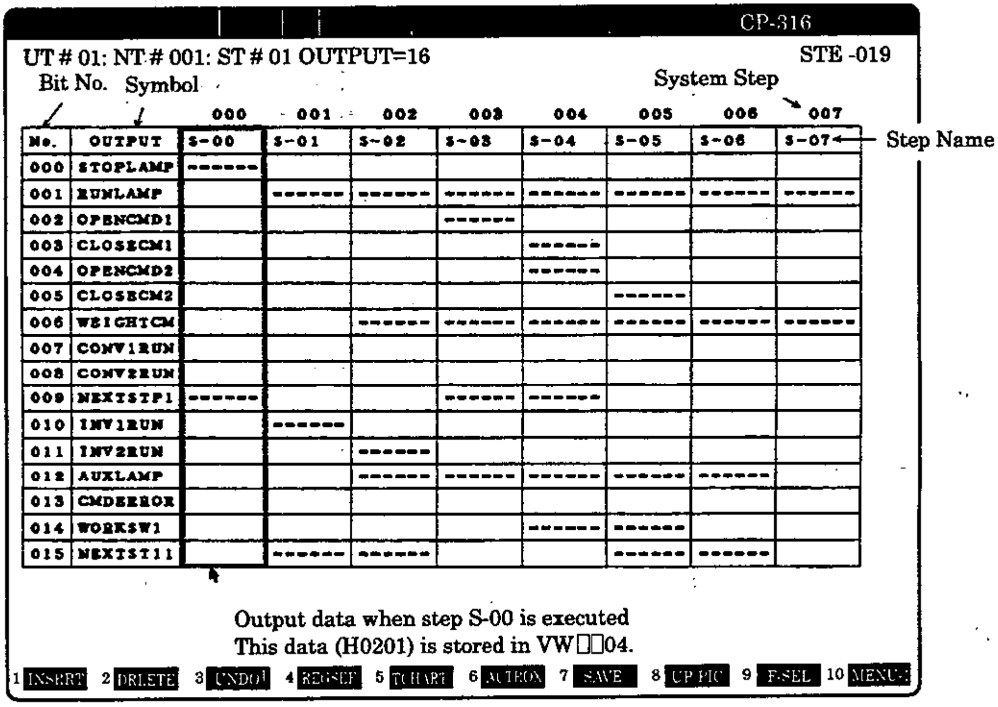


```
UT # 01: NT # 001: ST # 01 STEP. REG-DW00000                                  STEP -019

1 001 ABOX          S-00
1 001 ⊦ 00000                                                                  ⇒ MW13000
1 003 ⊦ 01000                                                                  ⇒ MW13001
1 005 SBOX          S-01
1 006 ⊦ MW13001     + 01000     × 02000     + 03000                            ⇒ MW13002
1 011 ABOX          S-02
1 012 | MB004900                                                                 MB007700 |
1 014 | MB004901                                                                 MB007701 |
1 016 | MB004902                                                                 MB007702 |
1 018 AEND
```

5.6 SFC Output Definition Time Chart

The SFC Output Definition Time Chart is used to designate the output data for each SFC step in time chart form. The output data that are designated at the SFC Output Definition Time Chart ar output to the SFC system operation registers (VW□□□ 04 to VW□□□ 07, VW□□□ 26 to VW□□□ 29 upon execution of the SFC program. The output data (VW□□□ 04 to VW□□□ 07, VW□□□ 26 t VW□□□ 29) are cleared to 0 before SFC execution, and updated after SFC execution. Therefore, th output data can not be referenced inside the Action Box. The following items should be set in the tim chart.

- **Step name**
  Each step name is displayed in each column.
- **Number of output points**
  Can be designated in multiples of 16 (max. = 128).
- **Output name**
  Can be specified with 8 or less alphanumeric characters.
  This is used as a comment.



UT # 01: NT # 001: ST # 01 OUTPUT=16 STE -019

		000	001	002	003	004	005	006	007
No.	OUTPUT	S-00	S-01	S-02	S-03	S-04	S-05	S-06	S-07
000	STOPLAMP	------							
001	RUNLAMP		------	------	------	------	------	------	------
002	OPENCMD1				------				
003	CLOSECM1					------			
004	OPENCMD2					------			
005	CLOSECM2						------		
006	WEIGHTCM			------	------	------	------	------	------
007	CONV1RUN								
008	CONV2RUN								
009	NEXTSTP1	------			------	------			
010	INV1RUN		------						
011	INV2RUN			------					
012	AUXLAMP			------	------	------	------	------	
013	CMDERROR								
014	WORKSW1					------	------		
015	NEXTST11		------	------			------	------	

Output data when step S-00 is executed
This data (H0201) is stored in VW□□04.

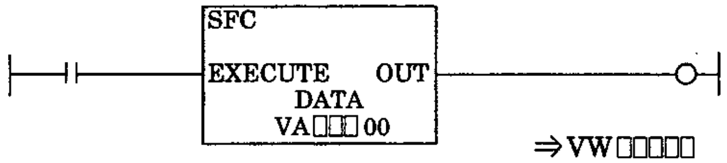
.7 Step Name Designation Method

The user may designate step names freely as long as they are within 6 alphanumeric characters in length and start with a character from "A" to "Z". However, use the following designating method if the user step number of a specific step name is to be taken out.

Step Name Designation Method

Step name : S nnnnn
- nnnnn → User step number (1 to 32767)*
- S → The initial character must be an "S."

(*) The user step number is a designated number given to a step by the user. It is not the number managed by the system.
(Examples) S0001, S0002, S0100, etc.



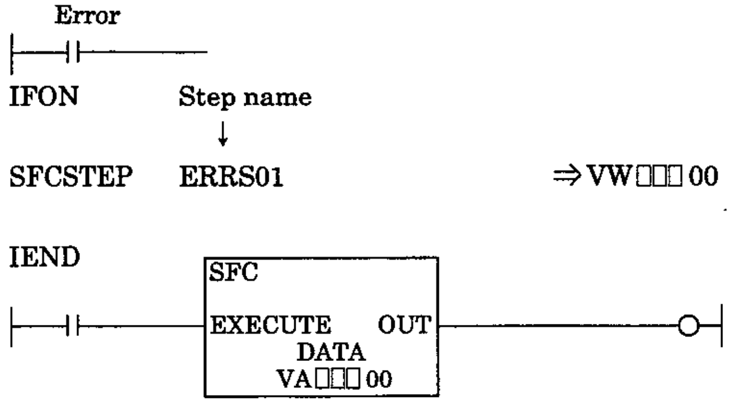
Execution of an SFC program.

The user step number of the currently executed SFC step is stored. When executing a multi-token, the user step number of the last executed SFC step is stored.

For step names designated by another method than the one above, the user step number becomes "0." In this case, a user step number which corresponds to the step name is not taken out.

.8 Taking Out System Step Nos.

The SFC controls the execution steps with the system step numbers that the system assigns automatically. In order to change the SFC execution forcibly to another step, take out the system step No. and change the execution step.
If an execution step is to be changed forcibly, such as in forced execution of a error processing sequence, the program is prepared using the SFCSTEP instruction . The SFCSTEP instruction takes out the system step number assigned to the step name. A program example of a error processing sequence is shown below.



When an error occurs:
Change the execution step forcibly by taking out the system step number of ERRS01 (error processing step) and storing it into the system step current value register (VW□□□ 00).

NOTE
1. If forced transition is to be performed, a timer transition condition cannot be used as a transition condition for the step which is the destination of transition.
2. Do not execute forced transition of an execution step from a step located within a multi-token structure.

5.9 Precautions upon Preparation of an SFC Program

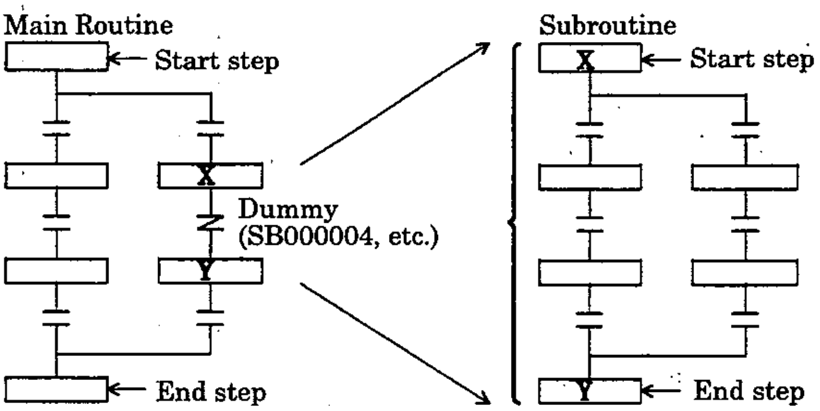
Note the precautions shown in Table 5.3 upon preparing an SFC program.

Table 5.3 Precautions upon Preparation of an SFC Program

Precaution	See Section
Only one SFC program can be programmed in one DWG.	—
The maximum number of steps in an SFC program is 500.	—
A branching or converging connection cannot be designated below and above one transition condition.	5.9.1
A convergence point must be provided if a multi-token structure is branched.	5.9.2
The number of branches in a multi-token structure must be 6 or less. [One cannot prepare a plurality of subroutines which have the same start step name and which contain a multi-token structure.]	5.9.3
A subroutine containing a multi-token structure cannot be called from within a multi-token block.	5.9.4
A subroutine containing a multi-token structure cannot be called from with a single-token block unless the conditions for subroutines are satisfied.	5.9.4
Subroutines may only be nested up to 4 times (depth of the macro) in each step in a single-token or multi-token block.	5.9.4
Jumping to a step in the middle of another block cannot be performed from a step inside a single-token or multi-token block.	5.9.4
The timer transition instruction cannot be used in a subroutine called from a multi-token structure.	5.9.4
The same step name cannot be used in different blocks.	5.9.5

Subroutine (Macro)

In cases where a step leads to more than 6 branches, the series of steps may be taken out and newly programmed as a separate block by assigning a representative step to the main routine. Such a block is called a subroutine (macro).



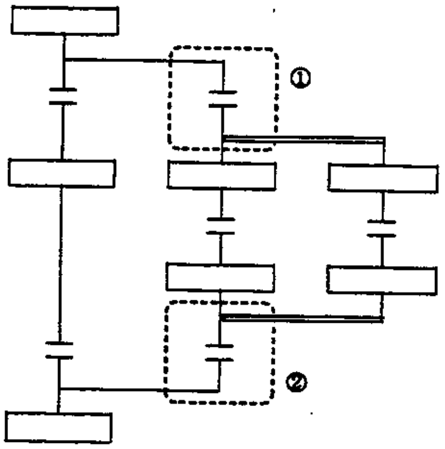
Block

A series of steps from a start step to an end step is called a block.

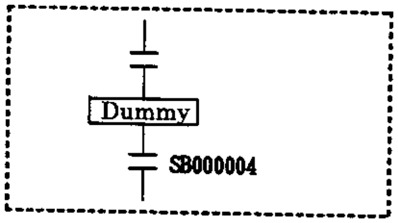
5.9.1 Restrictions concerning Branching and Converging Connections

A single-token or multi-token branching or converging connection cannot be designated below and above one transition condition. If branching and converging connections are not designated correctly, the program cannot be written in. Examples of restrictions concerning branching and converging connections and correct programming methods are shown below.

(Example 1)



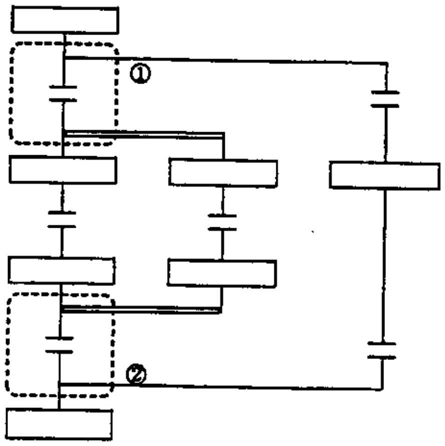
Correct programming



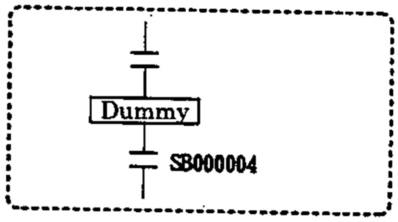
① Above : return point of a single-token structure
Below : branching point of a multi-token structure

② Above : convergence point of a multi-token structure
Below : convergence point of a single-token structure

(Example 2)



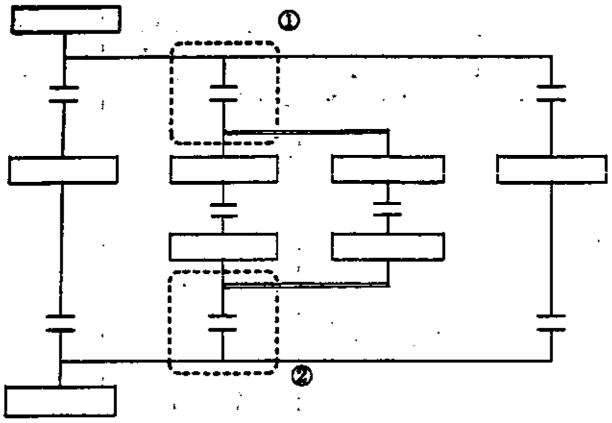
Correct programming



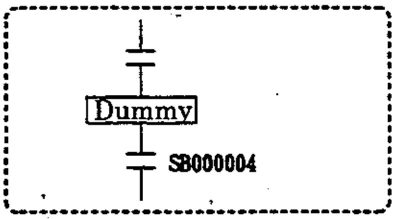
① Above : branching point of a single-token structure
Below : branching point of a multi-token structure

② Above : convergence point of a multi-token structure
Below : convergence point of a single-token structure

(Example 3)



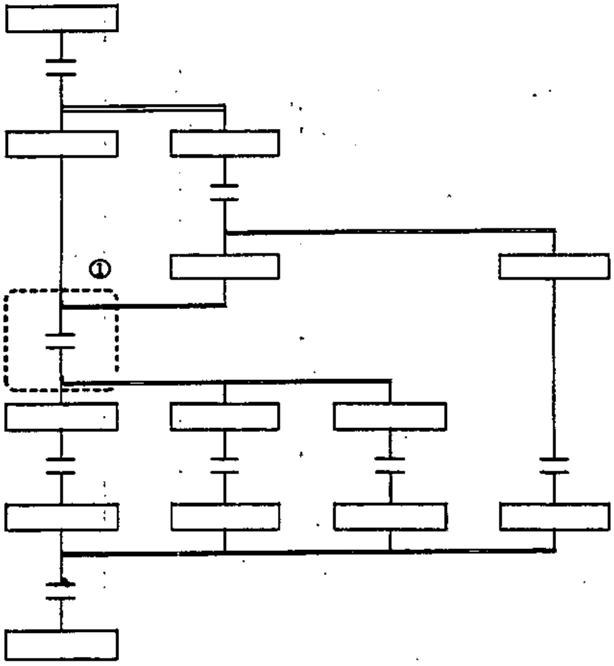
Correct programming



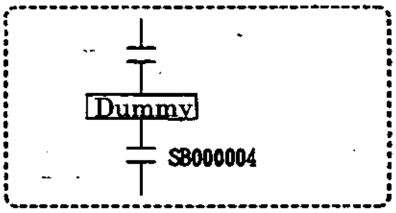
① Above: branching point of a single-token structure
Below: branching point of a multi-token structure

② Above : convergence point of a multi-token structure
Below : convergence point of a single-token structure

(Example 4)



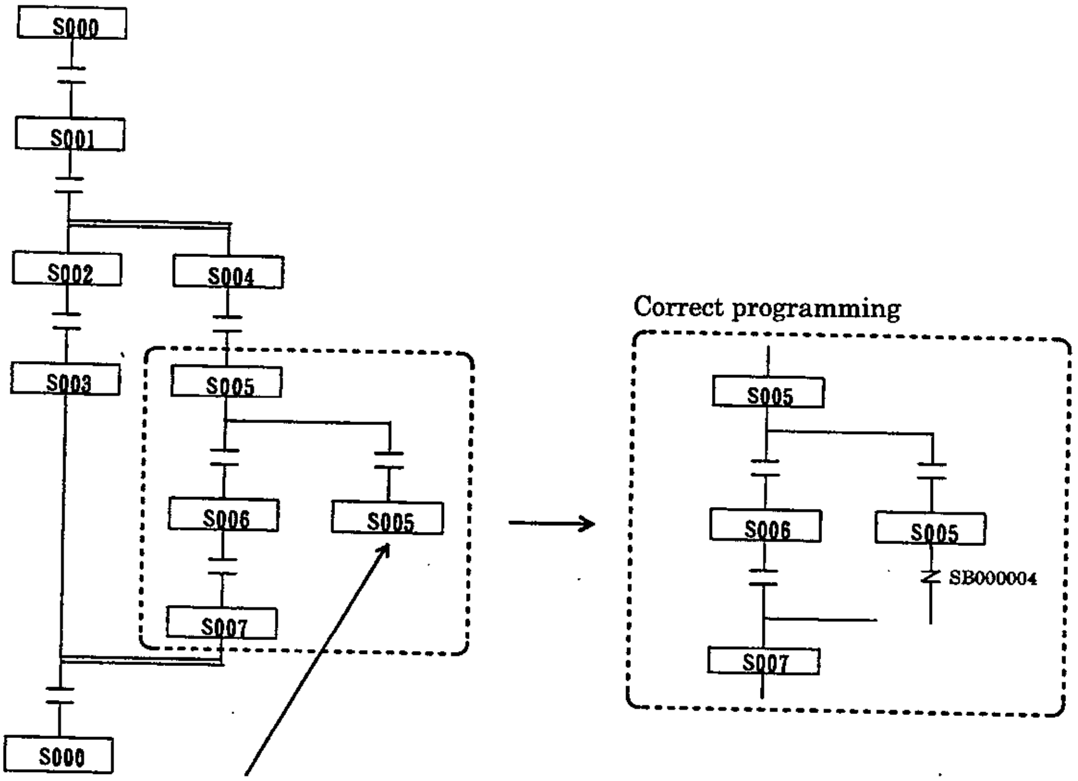
Correct programming



① Above : convergence point of a multi-token structure
Below : branching point of multi-token structure

5.9.2 Restriction concerning Branching and Converging Connections in a Multi-Token Structure

A convergence point must be provided if a multi-token structure is branched.
If branching and converging connections are not designated correctly, the program cannot be written in.



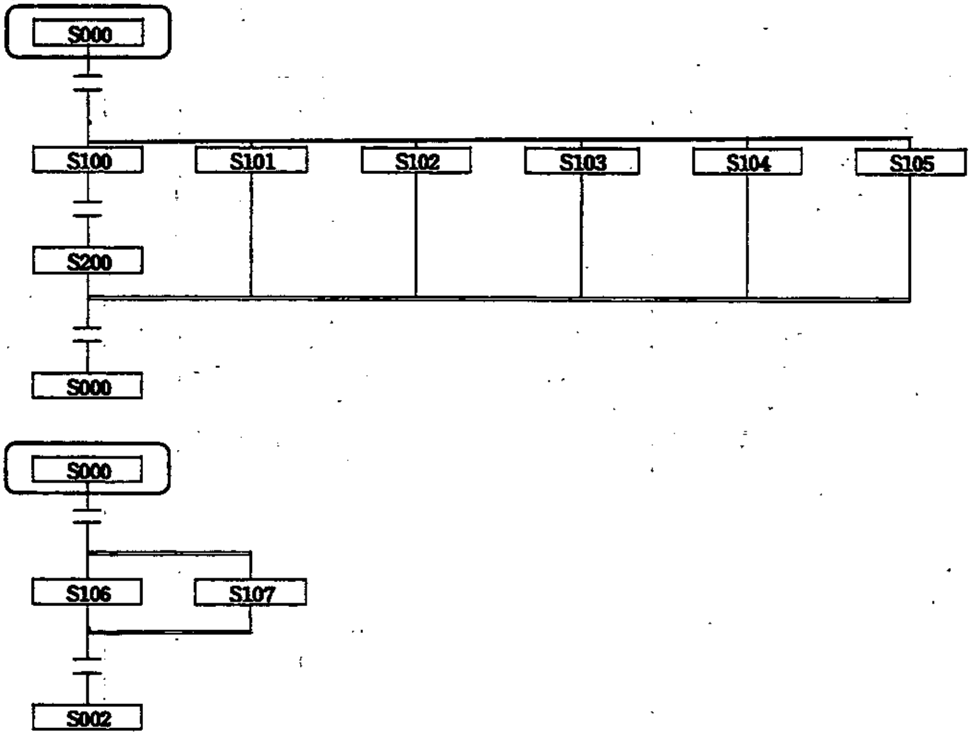
The multi-token structure remains branched since a step is set as an end step within a branch of a multi-token structure.

5.9.3 Restriction of the Number of Branches in a Multi-Token Structure

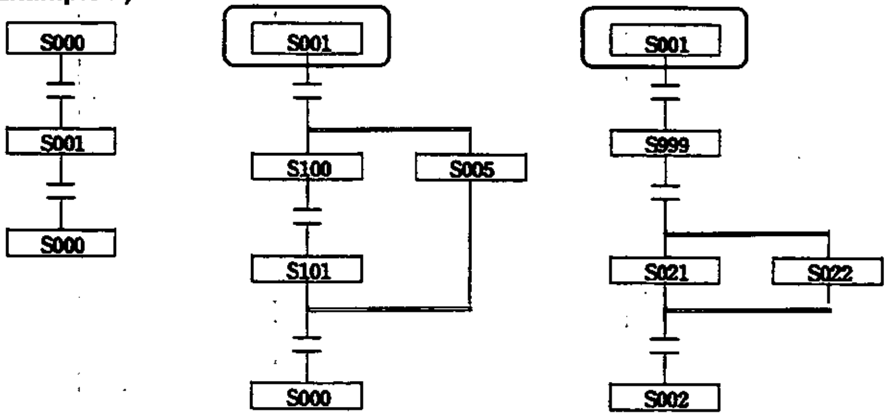
If there are 6 or more branches in one block in a single-token structure, the block may be divided i two to prepare the program. However, such a program cannot be prepared in the case of a multi-toke structure.

The maximum number of steps that can be executed parallel in a multi-token structure is 6. A progra with more than 6 branches will therefore be erroneous. A program will also be erroneous if there ar a plurality of blocks having the same start step name and containing a multi-token structure (se Examples 1 and 2). The program cannot be written in such cases. Change the program so that th number of parallel executed steps will be 6 or less. There are no restrictions in the number of branche in the case of a single-token structure.

(Example 1)



(Example 2)



9.4 Restrictions concerning Subroutines

Several conditions, which depend on whether the calling source of the subroutine (main routine) and the subroutine itself are a single-token structure or a multi-token structure, must be satisfied when preparing a subroutine in an SFC program. The program cannot be written in unless such conditions are satisfied.

Subroutine Main routine	Single-token block	Multi-token block
Single-token block	See ①.	See ②.
Multi-token block	See ③.	See ④.

① When calling a subroutine with a single-token block from a single-token block

- A compose error will occur if the following conditions are not satisfied

(Conditions)

1. Subroutines must not be nested more than 4 times.
2. Jumping must not be performed to a step in the middle of the subroutine.

② When calling a subroutine with a multi-token block from a single-token block

- A compose error will occur if conditions 1 and 2 below are not satisfied.
- A compile error will occur if condition 3 below is not satisfied.

(Conditions)

1. Subroutines must not be nested more than 4 times.
2. Jumping must not be performed to a step in the middle of the subroutine.
3. The subroutine side must not be branched immediately into a multi-token block.

③ When calling a subroutine with a single-token block from a multi-token block

- Compose error will occur if conditions 1 and 2 below are not satisfied.
- "WARNING" is issued if condition 3 below is not satisfied.

(Conditions)

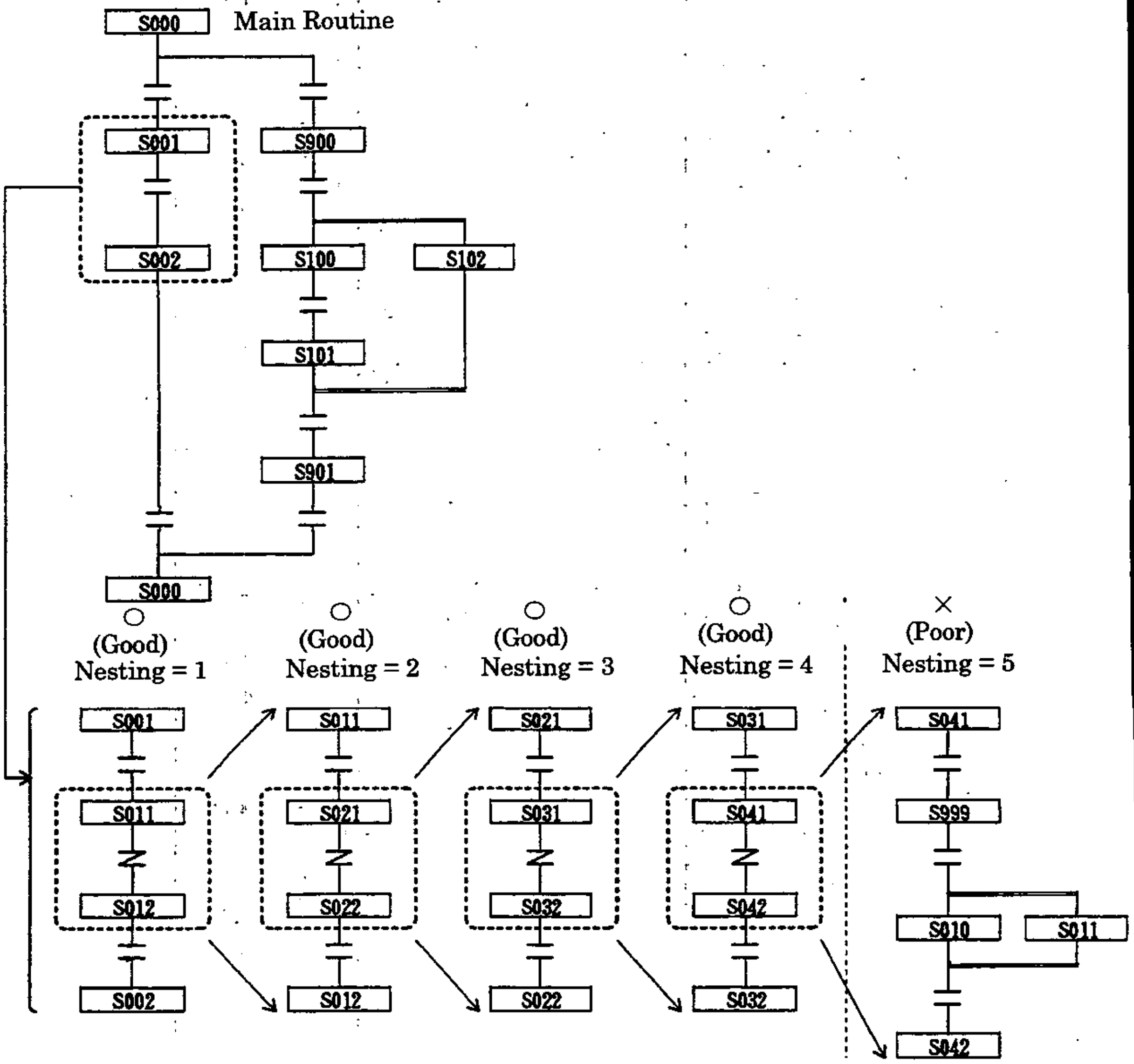
1. Subroutines must not be nested more than 4 times.
2. Jumping must not be performed to a step in the middle of the subroutine.
3. A timer transition instruction must not be used inside the subroutine.

④ When calling a subroutine with a multi-token block from a multi-token block

- Compose error will occur.

(1) Restrictions concerning Nesting (Depth of Macro)

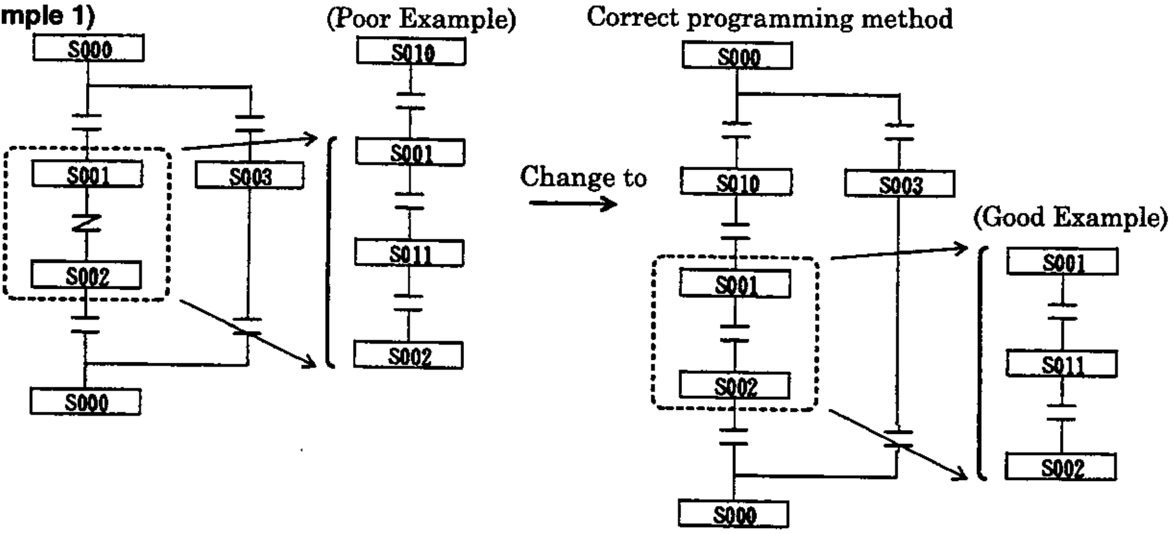
Subroutines can only be nested up to 4 times (depth of macro). Prepare the program so th
subroutines will be nested only 4 times or less.



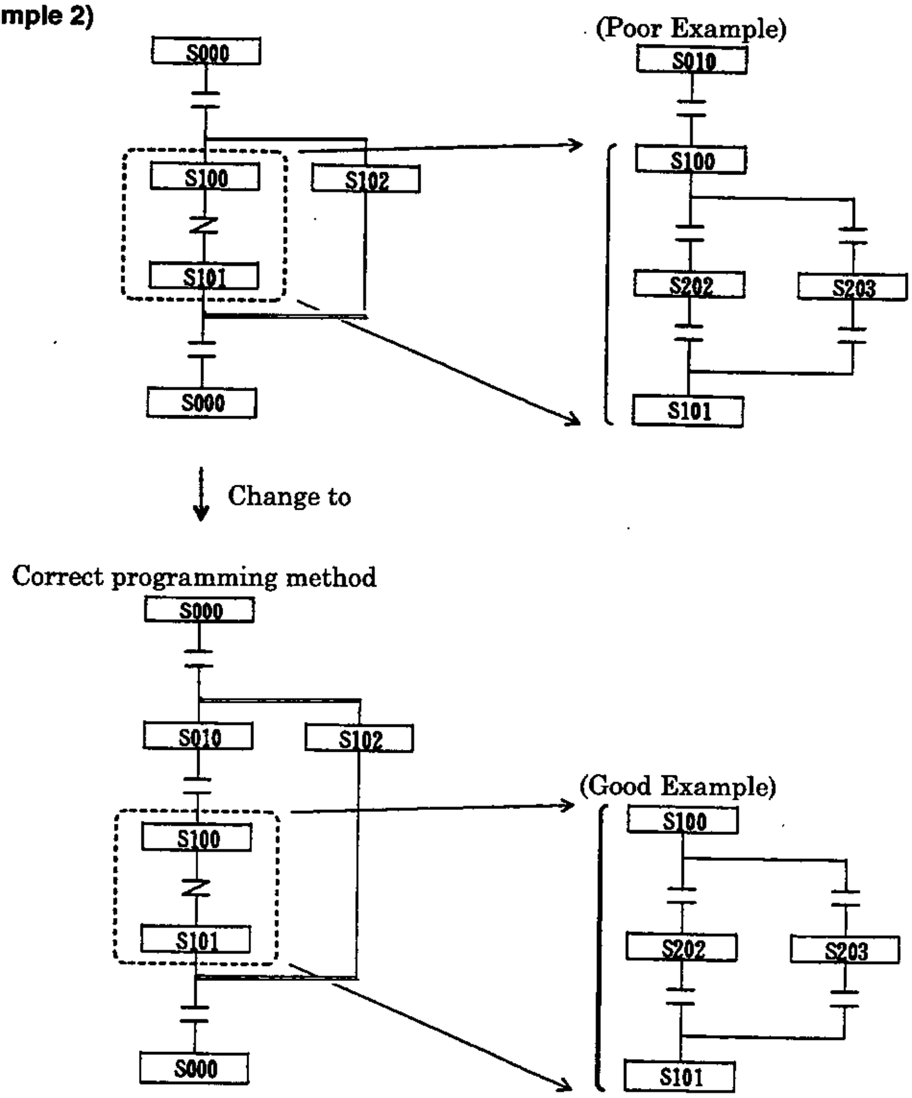
(2) Restrictions concerning Jumping

Programs, in which jumping is performed to a step in the middle of a subroutine as shown below, cannot be prepared. Shown below are examples of restrictions concerning jumping and correct programming methods.

(Example 1)

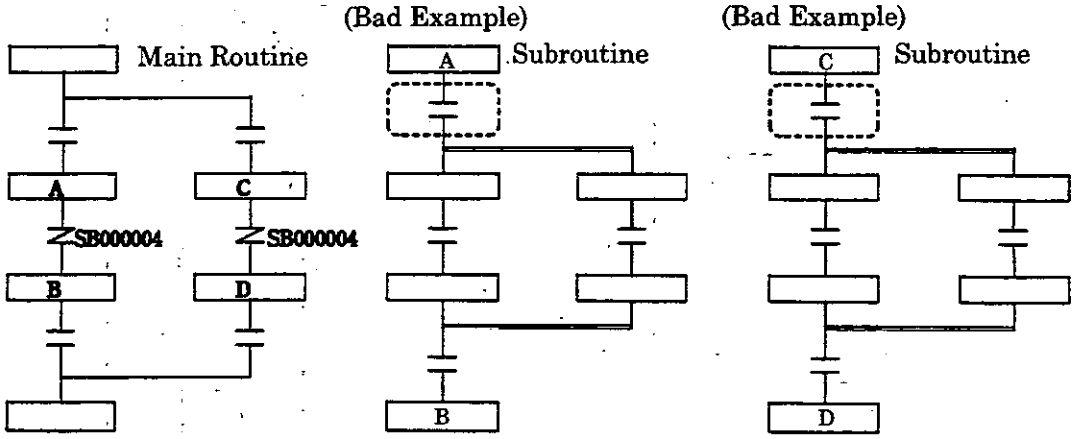


(Example 2)

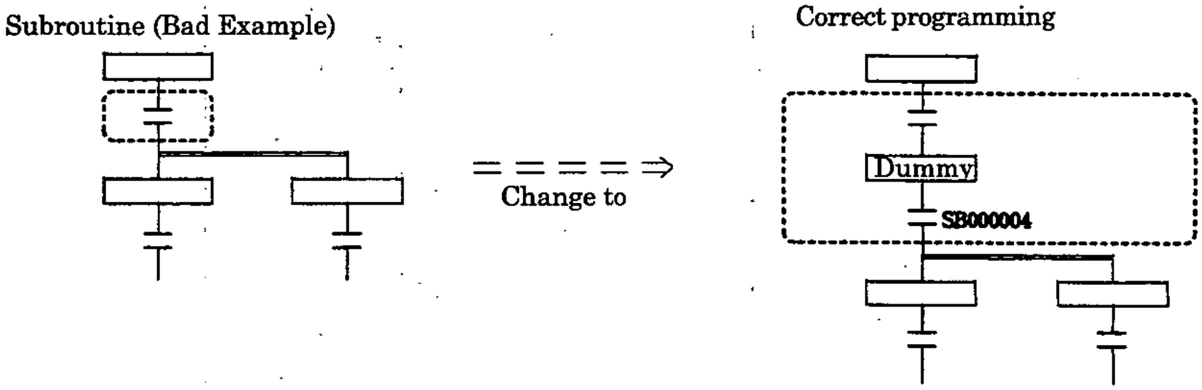


(3) Restrictions concerning Branching

Branching into a multi-token structure cannot be performed immediately after the start step o a subroutine called by a main routine. Shown below are examples of restrictions concernin branching and correct programming methods.


(Bad Example)

(Bad Example)

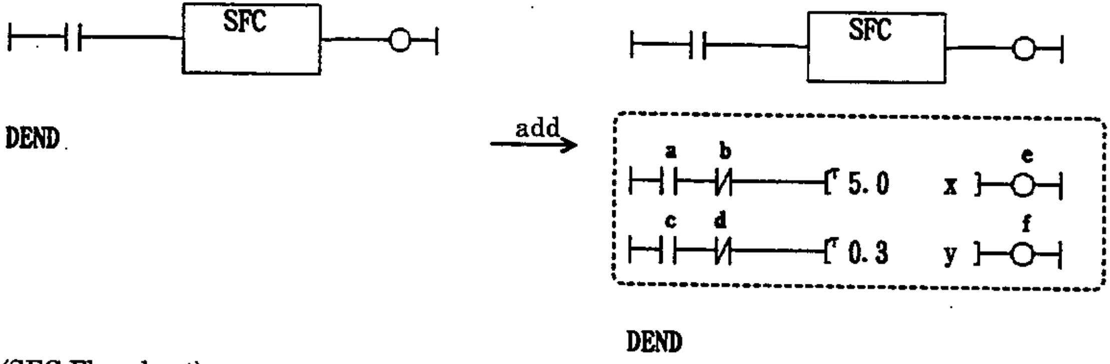

Subroutine (Bad Example)

Correct programming

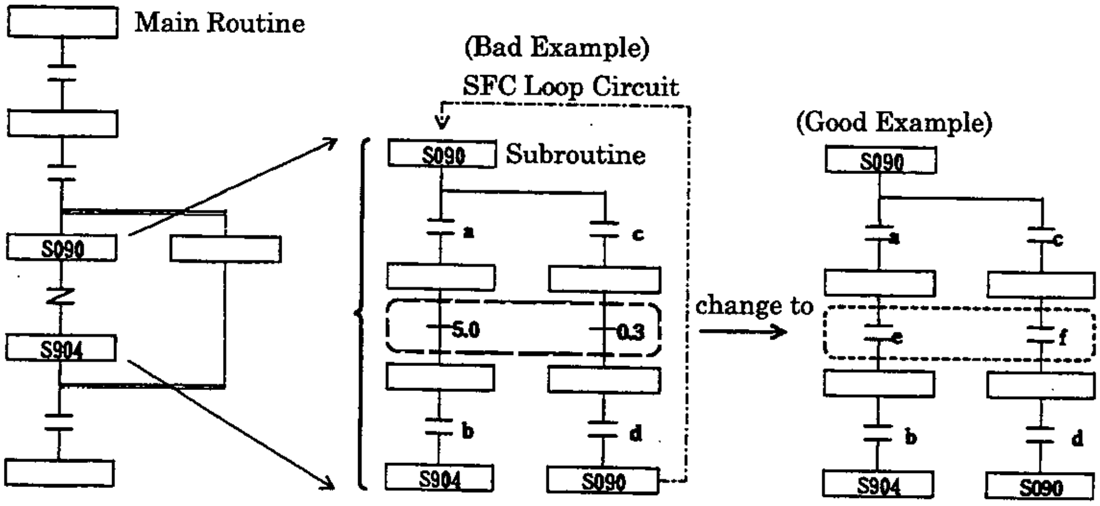
(4) Restrictions concerning the Timer Transition Condition Instruction

A timer transition instruction cannot be used in a subroutine that is called from a multi-token structure. If a timer is required, prepare a program in which an on-delay timer instruction is used outside the SFC (ladder program) and received by a coil and the coil is used as an NO contact transition instruction of the SFC. This programming method is shown below.

(Ladder Program)



(SFC Flowchart)



NOTE

The timer will operate correctly in the following exceptional cases.
However, ordinarily, change the program as shown above to avoid restrictions.

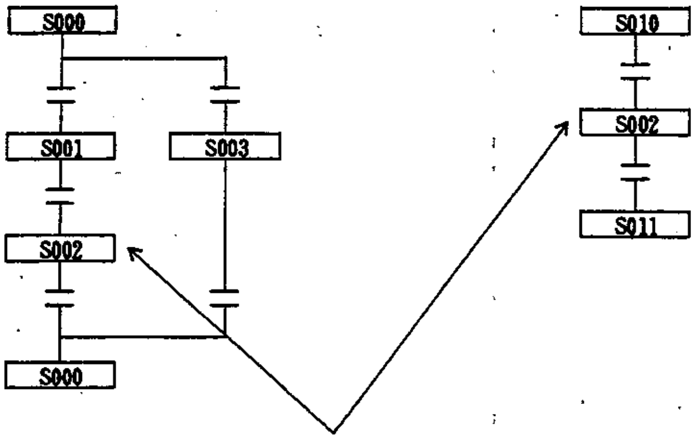
(1) When the following conditions are satisfied in one multi-token block:
(Conditions)
1. Only one subroutine is called from the multi-token structure, and
2. Only one timer transition instruction is used in the subroutine called, and
3. The timer transition instruction is not inside an SFC loop circuit.

(2) When the following conditions are satisfied when there are a plurality of subroutines called from a multi-token structure in one multi-token block:
(Conditions)
1. There is only one subroutine which uses a timer transition instruction, and
2. The timer transition instruction is not inside an SFC loop circuit.

5.9.5 Restrictions concerning Step Names

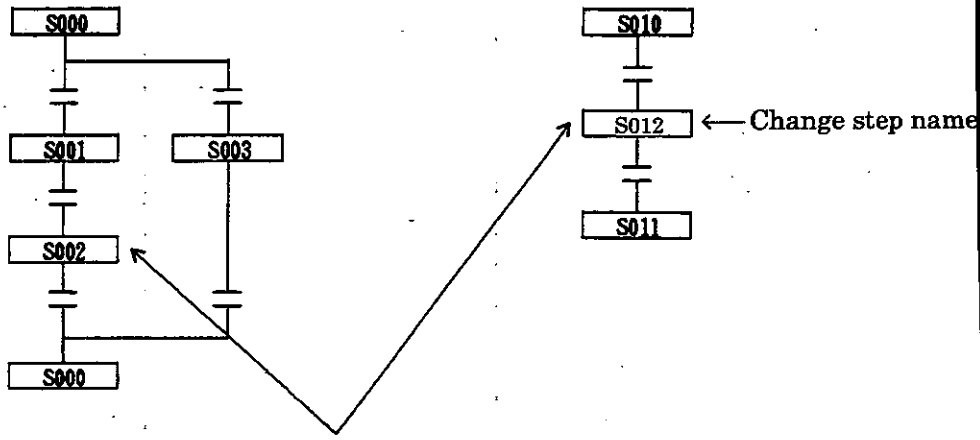
With the exception of the start step names and end step names in a macro, the same step nam cannot be used for different blocks. This condition applies in common to multi-token blocks an single-token blocks. Change the step names to prepare the program in such cases. A program canno be written in if the same step name is used.

(Bad Example)



Step name is overlapped.

change to



6 TABLE FORMAT PROGRAMMING

Table format programming methods, by which programs of a specific application are prepared in an FIF (fill in form) form by the use of tables, are described in this chapter. Constant tables, I/O conversion tables, interlock tables, part composition tables, and other various tables are made available. Some tables cannot be used with the programming device CP-717.

6.1 Types of Table Format Programs

As shown in Table 6.1, there are 6 types of table format programs. For functions, only the M registe constant table and the # register constant table can be used.

Table 6.1 Types of Table Format Programs

Name	Usage and Function	DWG	Function
Constant table (M register)	· Used for setting the various constant data, such as mechanical and electrical specifications of equipment, etc., that are used in common by different drawings. · Data names, symbols, units, and setting ranges can be designated	○	○
Constant table (# register)	· Used for setting various constant data, such as tension control parameters and position control parameters, that are used exclusively in a certain drawing. · Data names, symbols, units, and setting ranges can be designated.	○	○
I/O conversion table	· The I/O conversion processing parts of various processing programs may be prepared in a table. · Is provided with the scale conversion function and the bit signal conversion function. · Data names, symbols, units, and output conversion ranges can be designated.	○	×
Interlock table	· Used for preparing various types of interlocks. · A signal name and symbol can be designated for each input/output. · An interlock can be prepared as a combination of logical product (AND) and logical sum (OR) operations using NO contact and NC contact signals.	○	×
Part composition table	· Used to simultaneously prepare a plurality of circuits of a fixed pattern, such as solenoid circuits, accessory sequence circuits, etc. · Fixed-pattern circuits can be prepared and registered as standard software parts as necessary.	○	×
Constant Table (C register)	· Used in setting various types of data constants used in common on various drawings of mechanical and electrical sources of the equipment. · The data name, symbol, units, setting range, etc. can be designated.	×	×

○: can be used, × : cannot be used

NOTE
Make the table format programming on the programming device CP-717.

.2 Execution of Table Format Programs

Each table format program is executed with the XCALL instruction.

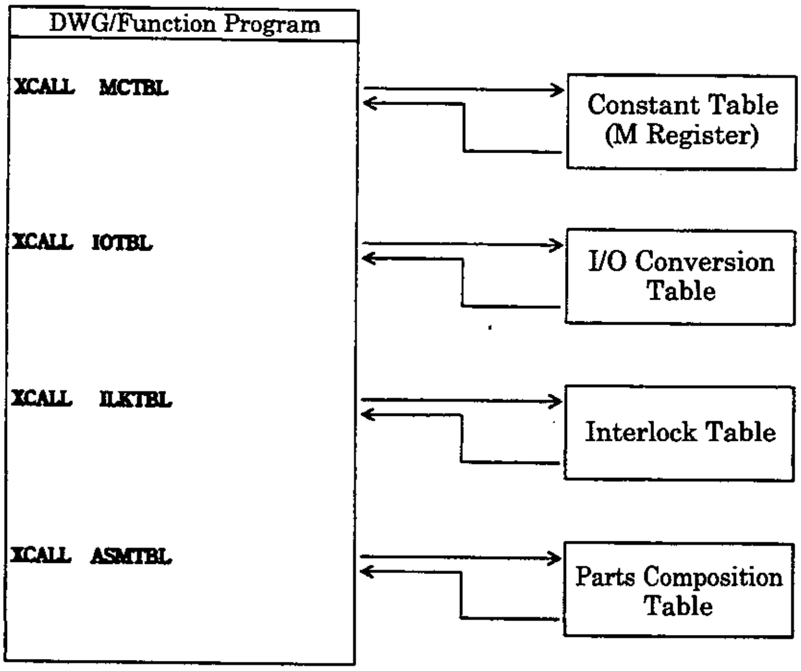


Table 6.1 Execution Method for Table Format Program

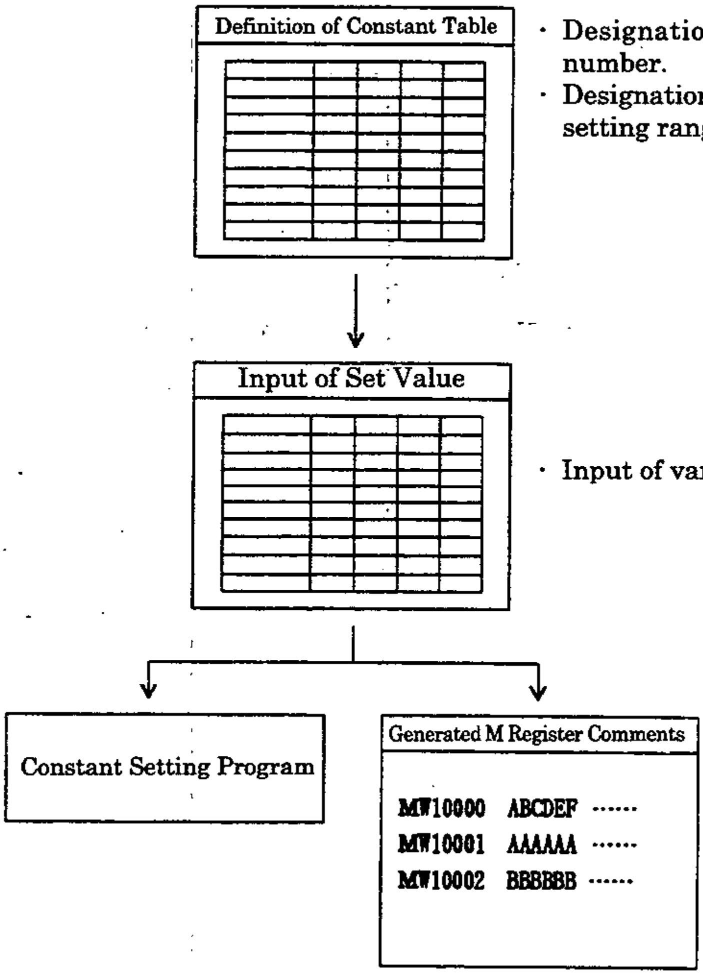
The set values for the constant table (# register) and the constant table (C register) are directly stored in # register and C register respectively.
Thus, it is not necessary to use the XCALL instruction for the constant table (# register) and the constant table (C register).

6.3 Constant Table (M Register)

The M register constant table is used for setting various constant data, such as mechanical and electri specifications of equipment, etc., that are used in common by different drawings.

6.3.1 Outline of the Constant Table (M Register)

To use the M register constant table, first a constant table is defined as shown in Fig. 6.2. The consta data are then set using the defined constant table.
When the constant table is stored, M register comments are prepared or renewed automatically accordi to the data name, symbol, unit, and register number of each row. These comments are used for comm display in the program screens and for comment printout upon printout of documents.



- Designation of the table name and drawing number.
- Designation of the data names, symbols, units, setting ranges, and storage addresses.

- Input of various set values

- M register comments are prepared renewed automatically when the register constant table is stored.

Fig. 6.2 Preparation of the M Register Constant Table

.3.2 Preparing the Constant Table (M Register)

(1) Defining the Constant Table (M Register)

The following items are set in defining the M register constant table. A maximum of 200 constants may be set.

① **Data Name**
Designate the data name of the constant.

② **Symbol**
Designate the symbol of the constant.

③ **Unit**
Designate the unit of the constant.

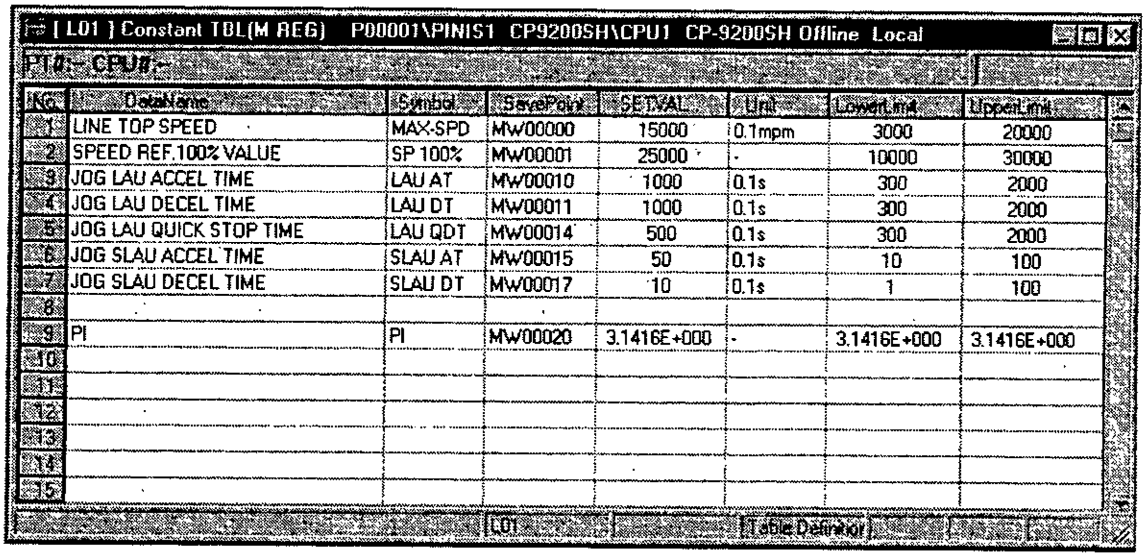
④ **Lower Limit**
Designate the lower input limit of the constant.

⑤ **Upper Limit**
Designate the upper input limit of the constant.

⑥ **Save Point**
Designate the M register into which the set values are stored.

(2) Inputs into the Constant Table (M Register)

The set value are input after the definition of the M register constant table has been completed.



[L01] Constant TBL(M REG)   P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local

PT#:- CPU#:-

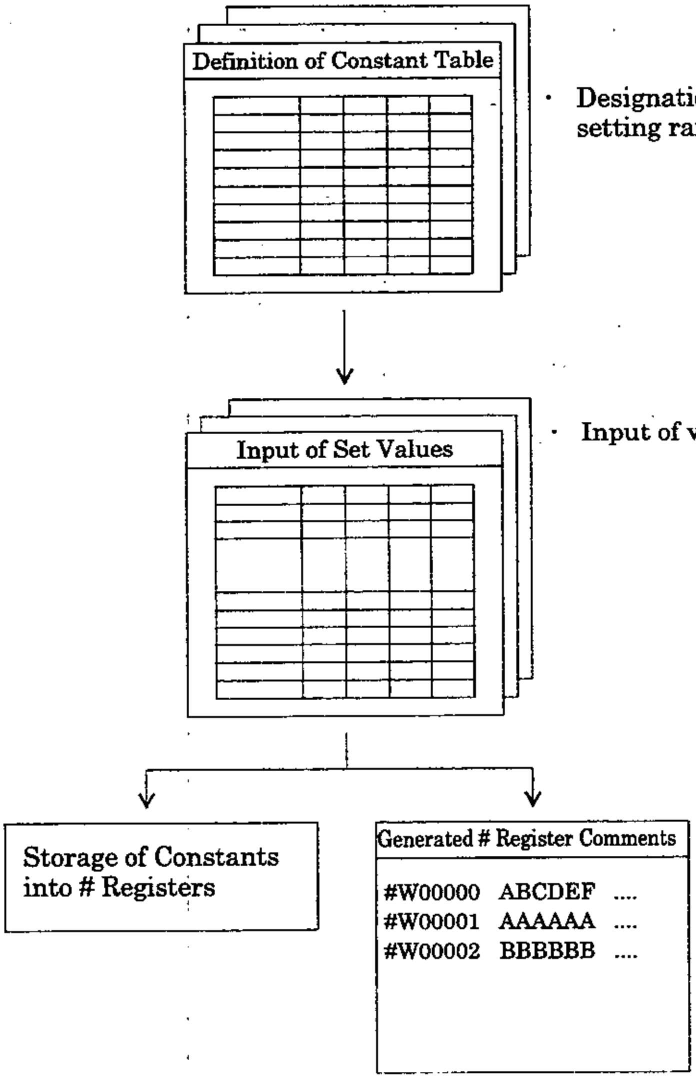
NO.	DataName	Symbol	SavePoint	SET VAL	Unit	LowerLimit	UpperLimit
1	LINE TOP SPEED	MAX-SPD	MW00000	15000	0.1mpm	3000	20000
2	SPEED REF.100% VALUE	SP 100%	MW00001	25000	-	10000	30000
3	JOG LAU ACCEL TIME	LAU AT	MW00010	1000	0.1s	300	2000
4	JOG LAU DECEL TIME	LAU DT	MW00011	1000	0.1s	300	2000
5	JOG LAU QUICK STOP TIME	LAU QDT	MW00014	500	0.1s	300	2000
6	JOG SLAU ACCEL TIME	SLAU AT	MW00015	50	0.1s	10	100
7	JOG SLAU DECEL TIME	SLAU DT	MW00017	10	0.1s	1	100
8							
9	PI	PI	MW00020	3.1416E+000	-	3.1416E+000	3.1416E+000
10							
11							
12							
13							
14							
15							

6.4 Constant Table (# Register)

The # register constant table is used for setting various constant data, such as tension control paramet and position control parameters, that are used exclusively in a certain drawing.

6.4.1 Outline of the Constant Table (# Register)

As shown in Fig. 6.3, the # register constant table is prepared in the same manner as the # regis constant table. A plurality of pages (up to 10 pages/DWG) can be used for the # register constant tab With the # register constant table, the settings of a plurality of pages are stored in the # registers of t designated drawing (DWG). Also, the # register comments are prepared when the settings are stor When the constant table is stored, # register comments are prepared or renewed automatically accordi to the data name, symbol, unit, and register number of each row. These comments are used for comme display in the program screens and for comment printout upon printout of documents.


Definition of Constant Table
Input of Set Values
Storage of Constants into # Registers
Generated # Register Comments
#W00000 ABCDEF ....
#W00001 AAAAAA ....
#W00002 BBBBBB ....

- Designation of the data names, symbols, units, setting ranges, and storage addresses.
- Input of various set values
- Generation of # register data and comments.

Fig. 6.3 Preparation of the # Register Constant Table

4.2 Preparing the Constant Table (# Register)

(1) Defining the Constant Table (# Register)

The following items are set in defining the # register constant table. A maximum of 100 constants may be set per page.

① **Data Name**
Designate the data name of the constant.

② **Symbol**
Designate the symbol of the constant.

③ **Unit**
Designate the unit of the constant.

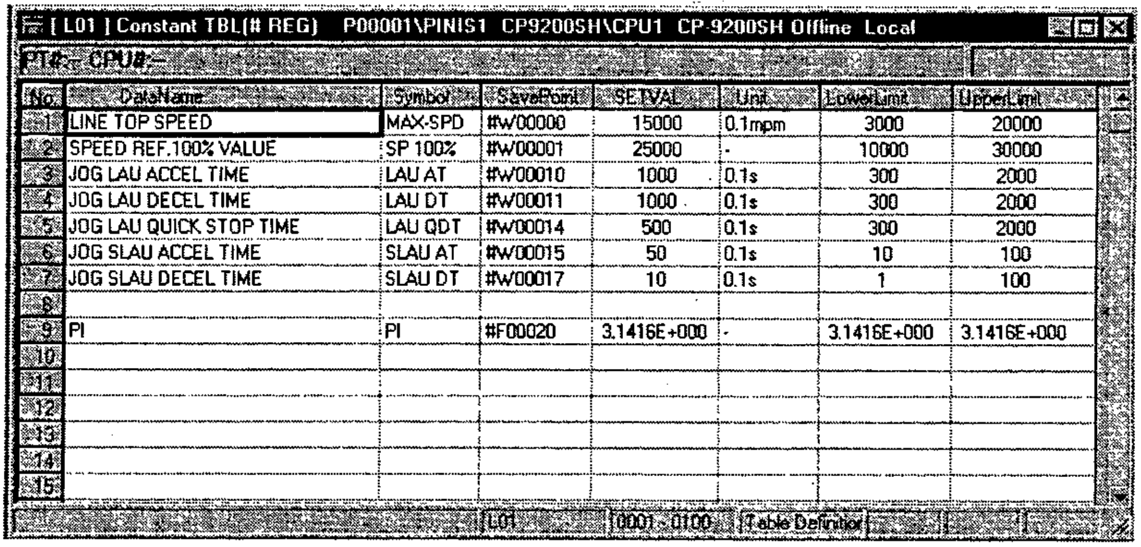
④ **Lower Limit**
Designate the lower input limit of the constant.

⑤ **Upper Limit**
Designate the upper input limit of the constant.

⑥ **Save Point**
Designate the # register into which the set values are stored.

(2) Inputs into the Constant Table (# Register)

The set values are input after the definition of the # register constant table has been completed. When the input of the set values has been completed, the set values of the various definition data are stored in the # registers of the designated drawings.



[L01] Constant TBL(# REG)   P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local

PT#:- CPU#:-

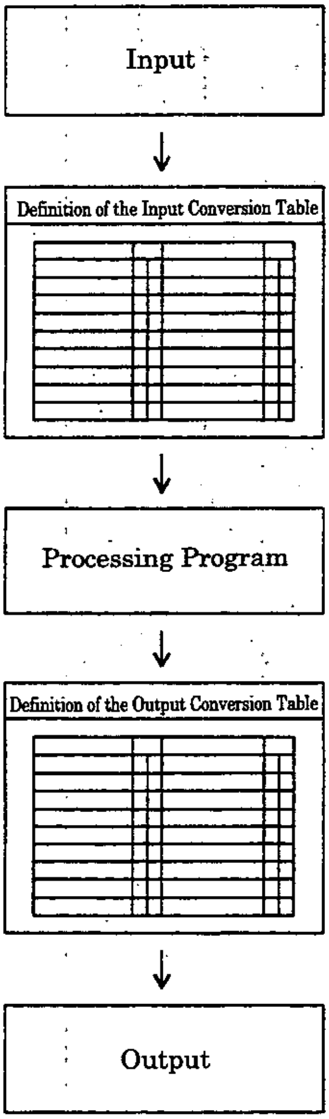
No.	DataName	Symbol	SavePoint	SETVAL	Unit	LowerLimit	UpperLimit
1	LINE TOP SPEED	MAX-SPD	#W00000	15000	0.1mpm	3000	20000
2	SPEED REF.100% VALUE	SP 100%	#W00001	25000	-	10000	30000
3	JOG LAU ACCEL TIME	LAU AT	#W00010	1000	0.1s	300	2000
4	JOG LAU DECEL TIME	LAU DT	#W00011	1000	0.1s	300	2000
5	JOG LAU QUICK STOP TIME	LAU QDT	#W00014	500	0.1s	300	2000
6	JOG SLAU ACCEL TIME	SLAU AT	#W00015	50	0.1s	10	100
7	JOG SLAU DECEL TIME	SLAU DT	#W00017	10	0.1s	1	100
8							
9	PI	PI	#F00020	3.1416E+000	-	3.1416E+000	3.1416E+000
10							
11							
12							
13							
14							
15							

6.5 I/O Conversion Table

The I/O conversion table enables the I/O conversion process of various processing programs to be prepar
as a table. Changes in I/O specifications can be made by simply changing definitions in the table.

6.5.1 Outline of the I/O Conversion Table

With the I/O conversion tables, tables for input conversion and tables for output conversions a
respectively prepared using different DWG's for each processing program.


Input
Definition of the Input Conversion Table
Processing Program
Definition of the Output Conversion Table
Output

- With the input conversion table, the input registers
registers) are usually used for the inputs and the M registe
that are used by the processing program, for the outputs

- With the output conversion table, the M registers, that a
used by the processing program, are used for the inputs a
the output registers (O registers) are used for the output

Fig. 6.4 Preparation of the I/O Conversion Table

6.5.2 Preparing the I/O Conversion Table

Scale conversion of numerical data and various signal conversions of bit signals can be designated with the I/O conversion table. Up to 1200 I/O conversions may be designated with one table (DWG).

(1) Scale Conversion Function

Addition, subtraction, multiplication, and division operations, that use immediate values and arbitrary registers, can be used as scale conversion functions. The following items should be set.

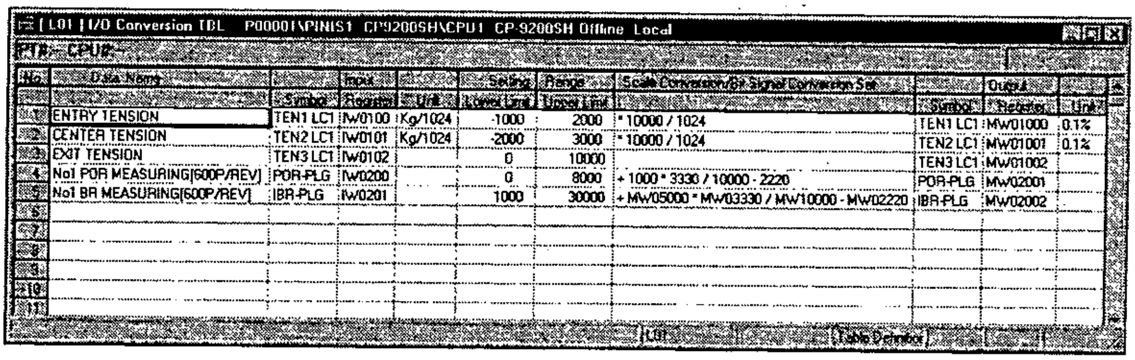
① **Data Name**
Designate the data name of the data to be converted.

② **Input**
Designate the register number, the unit, and the symbol of the input data at each row.

③ **Scale Conversion**
Designate addition, subtraction, multiplication, or division, that uses immediate values and arbitrary registers, for scale conversion.

④ **Setting Range**
Designate the upper and lower limits for the output.

⑤ **Output**
Designate the number of the register into which the conversion result is to be stored and the unit and the symbol of the output data at each row.



[L01] I/O Conversion TBL   P00001\PINIS1   CP9200SH\CPU1   CP-9200SH Offline   Local

PTR--CPU#--

No	Data Name	Input			Setting Range		Scale Conversion/Bit Signal Conversion Set	Output		
		Symbol	Register	Unit	Lower Limit	Upper Limit		Symbol	Register	Unit
1	ENTRY TENSION	TEN1 LC1	IW0100	Kg/1024	-1000	2000	* 10000 / 1024	TEN1 LC1	MW01000	0.1%
2	CENTER TENSION	TEN2 LC1	IW0101	Kg/1024	-2000	3000	* 10000 / 1024	TEN2 LC1	MW01001	0.1%
3	EXIT TENSION	TEN3 LC1	IW0102		0	10000		TEN3 LC1	MW01002	
4	No1 POR MEASURING[600P/REV]	POR-PLG	IW0200		0	8000	+ 1000 * 3330 / 10000 - 2220	POR-PLG	MW02001	
5	No1 BR MEASURING[600P/REV]	IBR-PLG	IW0201		1000	30000	+ MW05000 * MW03330 / MW10000 - MW02220	IBR-PLG	MW02002	
6										
7										
8										
9										
10										
11										

The I/O conversion designation of the 1st row of the above example realizes the same function as the following program.

```
 ├ IW0100    × 10000    ÷ 1024                              ⇒ MW01000
```

The I/O conversion designation of the 3rd row of the above example realizes the same function as the following program.

```
  ├ IW0102   < 00000
 [├ 00000  ] > 10000
 [├ 10000  ]                                                ⇒ MW01002
```

The I/O conversion designation of the 5th row of the above example realizes the same function as the following program.

```
  ├ IW0201   + MW05000 × MW03330 ÷ MW01000 − MW02220 < 01000
 [├ 01000  ] > 30000
 [├ 30000  ]                                                ⇒ MW02002
```

(2) Bit Signal Conversion Table

The 9 types of bit signal conversion shown in Table 6.2 can be designated.

Table 6.2 List of Conversion Symbols

Name	NO contact
NO contact	A ( )
NC contact	B ( )
Pulsed NO contact	PA ( )
Pulsed NC contact	PB ( )
NO contact timer	TA (□□□.□□)
NC contact timer	TB (□□□.□□)
Designated time pulse for NO contact	PTA (□□□.□□)
Designated time pulse for NC contact	PTB (□□□.□□)
NO contact chattering prevention	CTA (□□□.□□)

The following items should be set.

① **Data Name**

Designate the name of the signal to be converted.

② **Input**

Designate the relay number and the symbol for the input signal of each row.

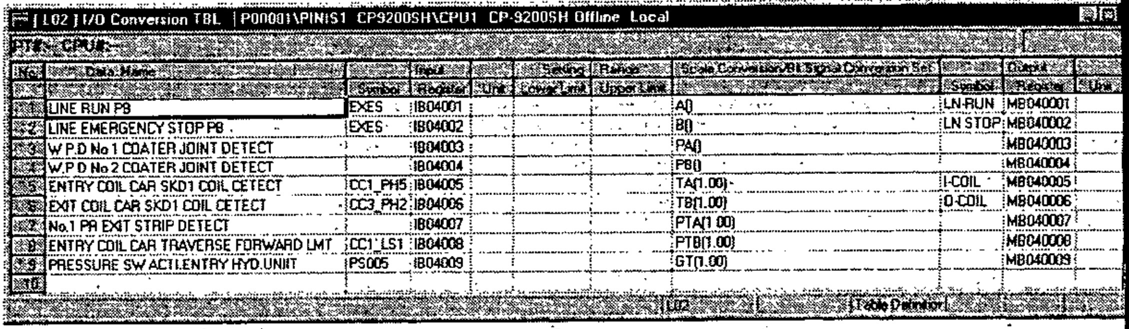
③ **Bit Signal Conversion Set**

Designate 9 types of bit signal conversion.

④ **Output**

Designate the number and symbol of the relay into which the conversion result is to be sto

for each row.



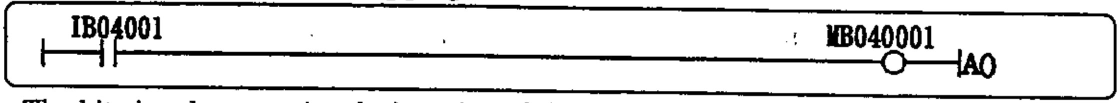
[L02] I/O Conversion TBL | P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local

PT#: CPU#:

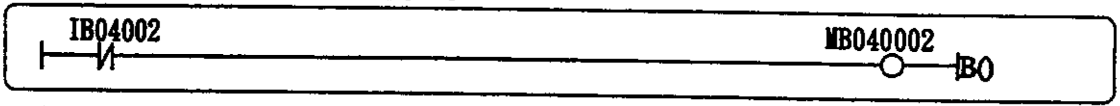
No.	Data Name	Input			Setting Range		Scale Conversion/Bit Signal Conversion Set	Output		
		Symbol	Register	Unit	Lower Limit	Upper Limit		Symbol	Register	Unit
1	LINE RUN PB	EXES	IB04001				A()	LN-RUN	MB040001	
2	LINE EMERGENCY STOP PB	EXES	IB04002				B()	LN STOP	MB040002	
3	W P.D No 1 COATER JOINT DETECT		IB04003				PA()		MB040003	
4	W.P D No 2 COATER JOINT DETECT		IB04004				PB()		MB040004	
5	ENTRY COIL CAR SKD1 COIL CETECT	CC1_PH5	IB04005				TA(1.00)	I-COIL	MB040005	
6	EXIT COIL CAR SKD1 COIL CETECT	CC3_PH2	IB04006				TB(1.00)	O-COIL	MB040006	
7	No.1 PR EXIT STRIP DETECT		IB04007				PTA(1.00)		MB040007	
8	ENTRY COIL CAR TRAVERSE FORWARD LMT	CC1_LS1	IB04008				PTB(1.00)		MB040008	
9	PRESSURE SW ACT!.ENTRY HYD.UNIIT	PS005	IB04009				GT(1.00)		MB040009	
10										

Equivalent Ladder Programs

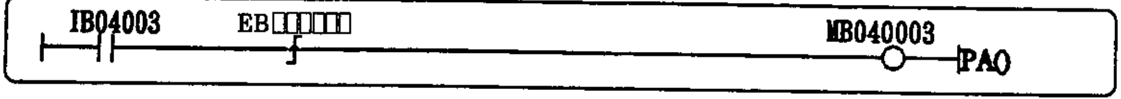
The bit signal conversion designation of the 1st row of the above example realizes the same function as the following program.



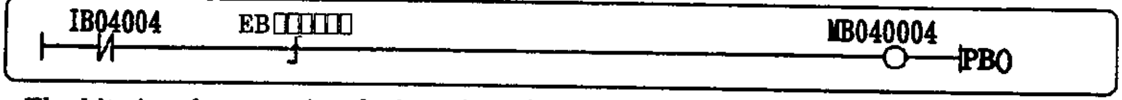
The bit signal conversion designation of the 2nd row of the above example realizes the same function as the following program.



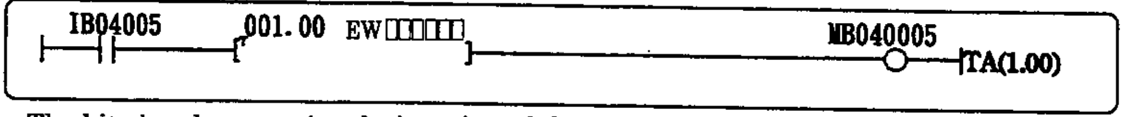
The bit signal conversion designation of the 3rd row of the above example realizes the same function as the following program.



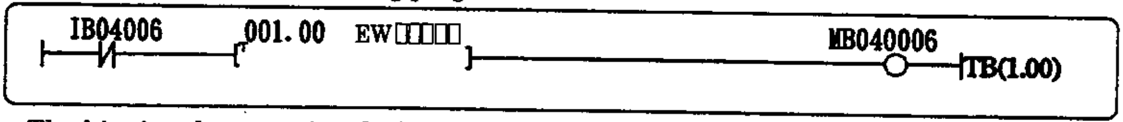
The bit signal conversion designation of the 4th row of the above example realizes the same function as the following program.



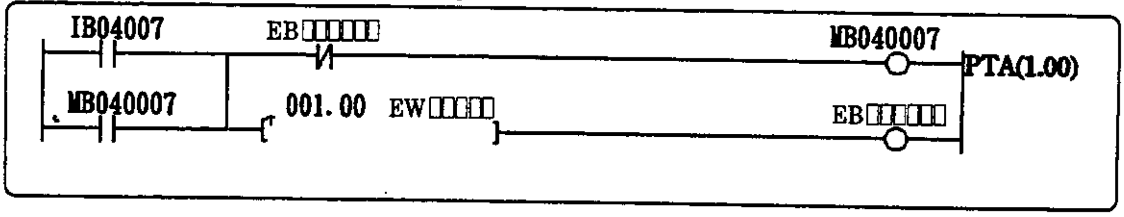
The bit signal conversion designation of the 5th row of the above example realizes the same function as the following program.



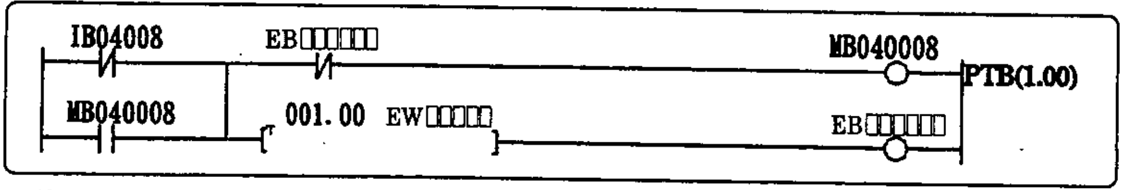
The bit signal conversion designation of the 6th row of the above example realizes the same function as the following program.



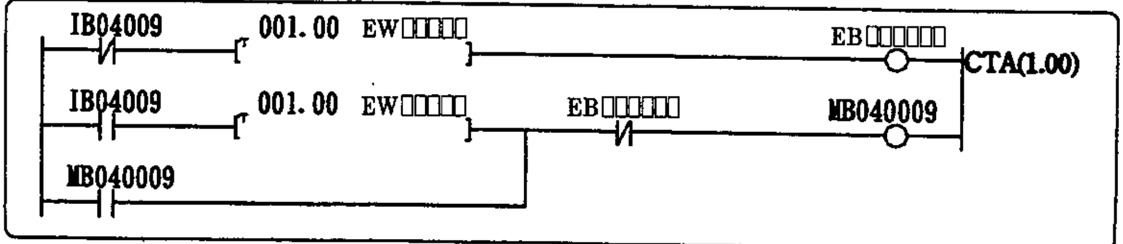
The bit signal conversion designation of the 7th row of the above example realizes the same function as the following program.



The bit signal conversion designation of the 8th row of the above example realizes the same function as the following program.



The bit signal conversion designation of the 9th row of the above example realizes the same function as the following program.



(Note) : The E registers are registers used by the controller. It is impossible for a user to directly read or write.

6.6 Interlock Table

The interlock table is used to prepare various interlocks, for starting conditions, running conditio etc. of devices, in table format.

6.6.1 Outline of the Interlock Table

As shown in Fig. 6.5, the interlock table is composed of one main interlock table and the correspond sub interlock tables. One sub interlock table may be set for one row of the main interlock table. The s interlock table is used to prepare specific input signals for the main interlock table. The main interl table may be divided into several blocks. The maximum number of blocks is 26 and each block handled as an independent interlock. When the interlock table is stored, comments for the regist (relays) are prepared or renewed automatically according to the data name, symbol, and register num (relay number) of each row. These comments are used for comment display in the program screens a for comment printout upon printout of documents.

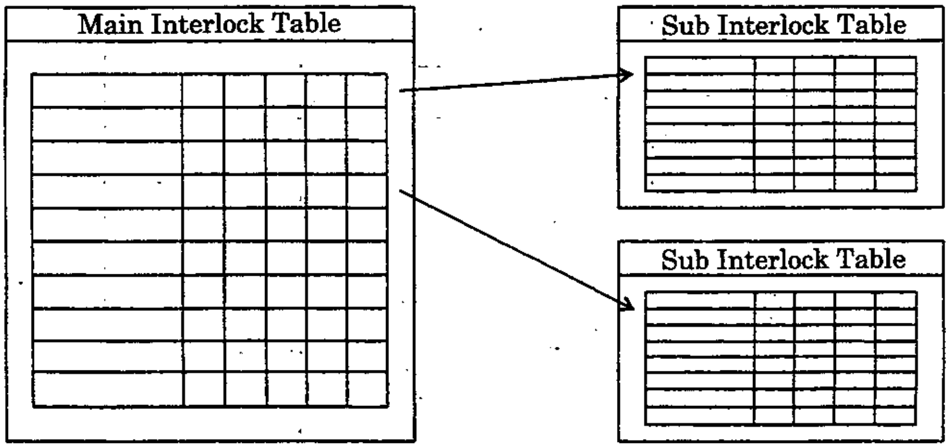


Fig. 6.5 Preparation of the Interlock Table

.2 Preparing the Interlock Table

Each interlock table (main or sub interlock table) is prepared in the same manner as follows. A maximum of 500 rows and 25 columns of data can be set.

① **Classification of the I/O signal:** This is designated according to the mode (M). The following 4 modes can be used.
- I : Designates a signal to be an input signal.
- S : Designates an output signal from a sub interlock table to be used as an input signal.
- O : Designates a signal to be an output signal.
- X : Designates the contact of an output signal to be used as an input (self-hold circuit).

② **Data Name**
Designate the name of the interlock condition to be input for each row.

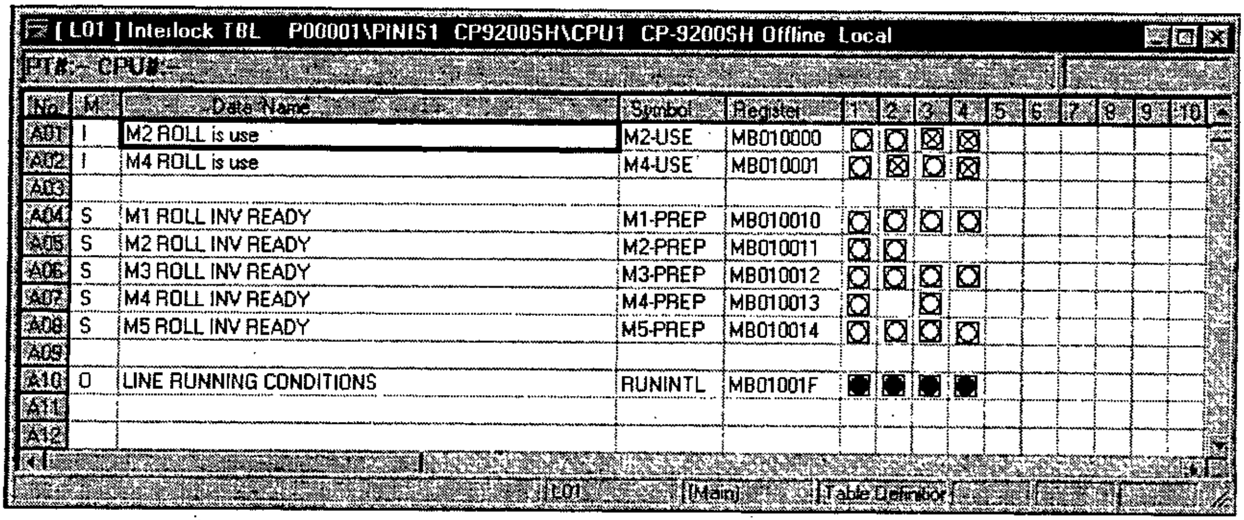
③ **Symbol**
Designate the symbol of the interlock condition to be input for each row.

④ **Register**
Designate the register number of the interlock condition to be input for each row.

⑤ **Interlock Input Condition**
For each input signal, designate the interlock condition, which is to be used as the condition for obtaining the logic product (AND) for each column. The NO contact condition (⧇) and the NC contact condition (⊠) can be used as interlock conditions.

⑥ **Interlock Input Condition**
For each output signal, designate (▣) the above mentioned interlock conditions to be used as conditions for obtaining the logic sum (OR) for the corresponding row.

The logical product (AND) of the input symbols, which were designated as the interlock conditions, is determined for each column and the output signal is prepared as the logic sum (OR) condition of the logical product results of the columns designated at each output signal row. Thus the following interlock table will be equivalent to the ladder program shown in the next page.

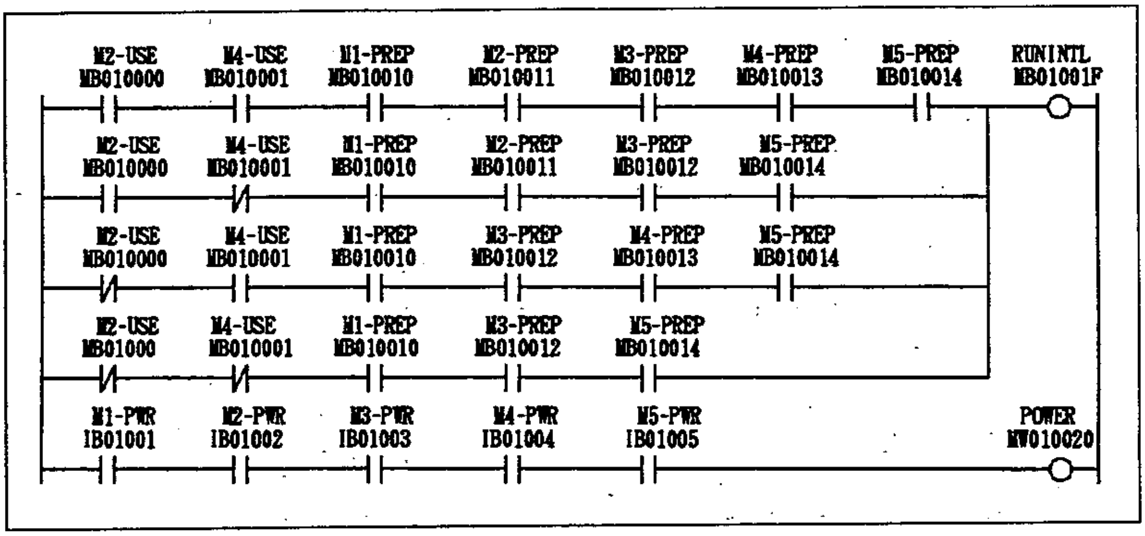


[L01] Interlock TBL P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local

PT#:- CPU#:-

No.	M	Data Name	Symbol	Register	1	2	3	4	5	6	7	8	9	10
A01	I	M2 ROLL is use	M2-USE	MB010000	⧇	⧇	⊠	⊠						
A02	I	M4 ROLL is use	M4-USE	MB010001	⧇	⊠	⧇	⊠						
A03														
A04	S	M1 ROLL INV READY	M1-PREP	MB010010	⧇	⧇	⧇	⧇						
A05	S	M2 ROLL INV READY	M2-PREP	MB010011	⧇	⧇								
A06	S	M3 ROLL INV READY	M3-PREP	MB010012	⧇	⧇	⧇	⧇						
A07	S	M4 ROLL INV READY	M4-PREP	MB010013	⧇		⧇							
A08	S	M5 ROLL INV READY	M5-PREP	MB010014	⧇	⧇	⧇	⧇						
A09														
A10	O	LINE RUNNING CONDITIONS	RUNINTL	MB01001F	▣	▣	▣	▣						
A11														
A12														

Equivalent Ladder Program

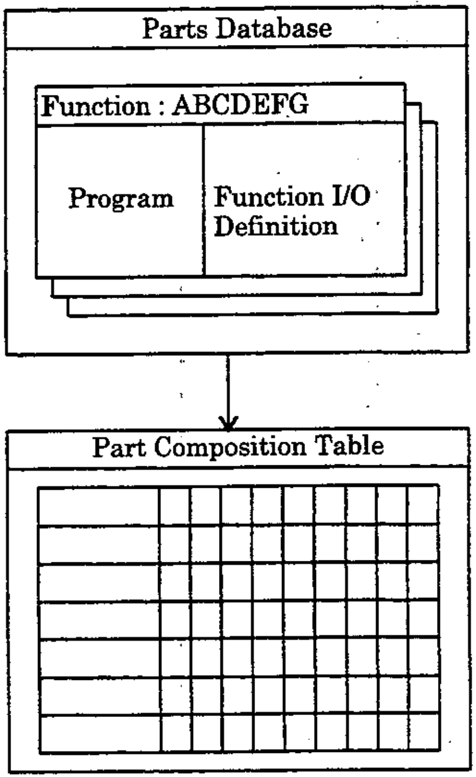


6.7 Part Composition Table

The part composition table is used to simultaneously prepare a plurality of circuits of a fixed patte such as solenoid circuits, accessory sequence circuits, etc.

6.7.1 Outline of the Part Composition Table

The part composition table is composed of functions, that are used as parts, and the part compositi table. The functions to be used as parts should be prepared before using them in the part compositi table.



- Each part is composed of the main body of the functi program and the function I/O definition.

- A plurality of circuits, which use the designated parts, a prepared simultaneously.
- Each circuit is prepared by defining the inputs and outpu of the circuit with the register numbers.

Fig 6.6 Preparation of the Part Composition Table

.7.2 Preparing the Part Composition Table

With the part composition table, a plurality of circuits with the same pattern can be prepared simultaneously using designated parts. In the part composition table, one row corresponds to one circuit and names, inputs, and outputs are designated for each row to prepare a plurality of circuits. The parts to be used can be designated for each row. The maximum number of inputs and the maximum number of outputs is designated by the user. A maximum of 100 circuits can be prepared.

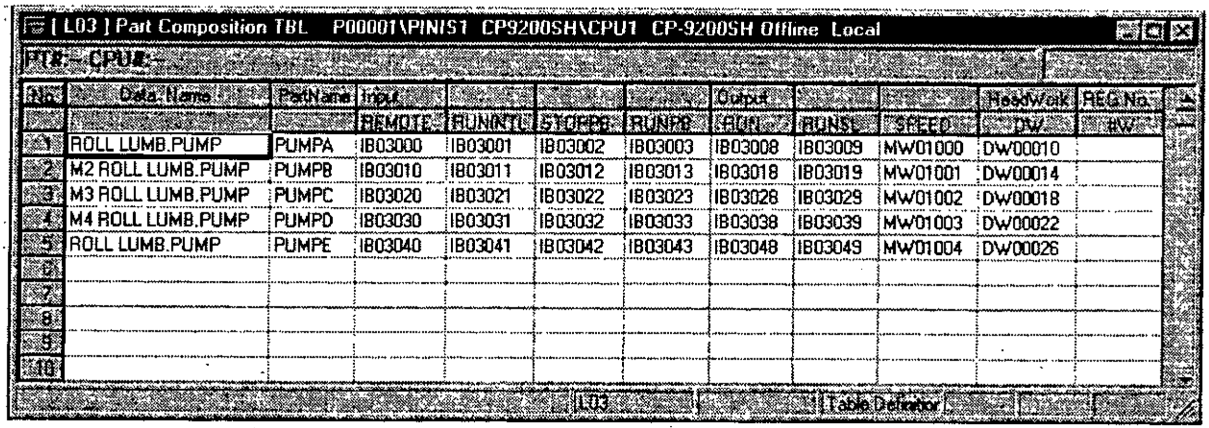
① **Data Name**
Designate the name of each circuit.

② **Part Name**
Designate the function symbol or user function name of the function to be used as a part.

③ **Input**
Use register numbers to designate the inputs of each circuit. The register whose number is designated here will provide the input to the user function.

④ **Output**
Use register numbers to designate the outputs of each circuit.

⑤ **Head Work**
Designate the number, in word form, of the D register or # register which is to be the head work register to be used for each circuit.


[L03] Part Composition TBL   P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local
PT#-- CPU#--
L03
Table Definition

No.	Data Name	PartName	Input				Output			HeadWork	REG No.
			REMOTE	RUNINTL	STOPPB	RUNPB	RUN	RUNSL	SPEED	DW	#W
1	ROLL LUMB.PUMP	PUMPA	IB03000	IB03001	IB03002	IB03003	IB03008	IB03009	MW01000	DW00010	
2	M2 ROLL LUMB.PUMP	PUMPB	IB03010	IB03011	IB03012	IB03013	IB03018	IB03019	MW01001	DW00014	
3	M3 ROLL LUMB.PUMP	PUMPC	IB03020	IB03021	IB03022	IB03023	IB03028	IB03029	MW01002	DW00018	
4	M4 ROLL LUMB.PUMP	PUMPD	IB03030	IB03031	IB03032	IB03033	IB03038	IB03039	MW01003	DW00022	
5	ROLL LUMB.PUMP	PUMPE	IB03040	IB03041	IB03042	IB03043	IB03048	IB03049	MW01004	DW00026	
6											
7											
8											
9											
10											

6.7.3 Preparing the Function Program for Parts

The parts (main bodies of function programs and function I/O definitions) to be used in a part compositi table should be prepared in advance. Although the preparation method is the same as that for ordina function programs, the following data are used for the input/output of parts and the work register.

Input of Parts

The inputs designated at the function I/O definition will be used as the inputs for the parts. Refer "Chapter 3 REGISTER MANAGEMENT METHOD" concerning the relationship between the inp definition for a function and the input variables (X registers) used in the function.

Output of Parts

The outputs designated at the function I/O definition will be used as the outputs for the parts. Re to "Chapter 3 REGISTER MANAGEMENT METHOD" concerning the relationship between t output definition for a function and the output variables (Y registers) used in the function.

Work Register

The Z register corresponds to the D register of a DWG and the # register corresponds to th register of a drawing and the sum of the head work register number of the part composition ta and the relative register number of that register is used as the number of the actual work regist

.8 Constant Table (C Register)

The C register constant table is used for setting various data constants common to all DWG such as equipment and manufactured sources. A maximum of 200 constant tables (C register) can be created.

.8.1 Outline of the Constant Table (C Register)

Multiple definitions of set values are stored in the C register by the constant table (C register).
Also, the C register comments are prepared at the same time the set values are stored.
When the constant table is stored, C register comments are prepared or renewed automatically according to the data name, symbol, unit, and register number of each row. These comments are used for comment display in the program screens and for comment printout upon printout of documents.

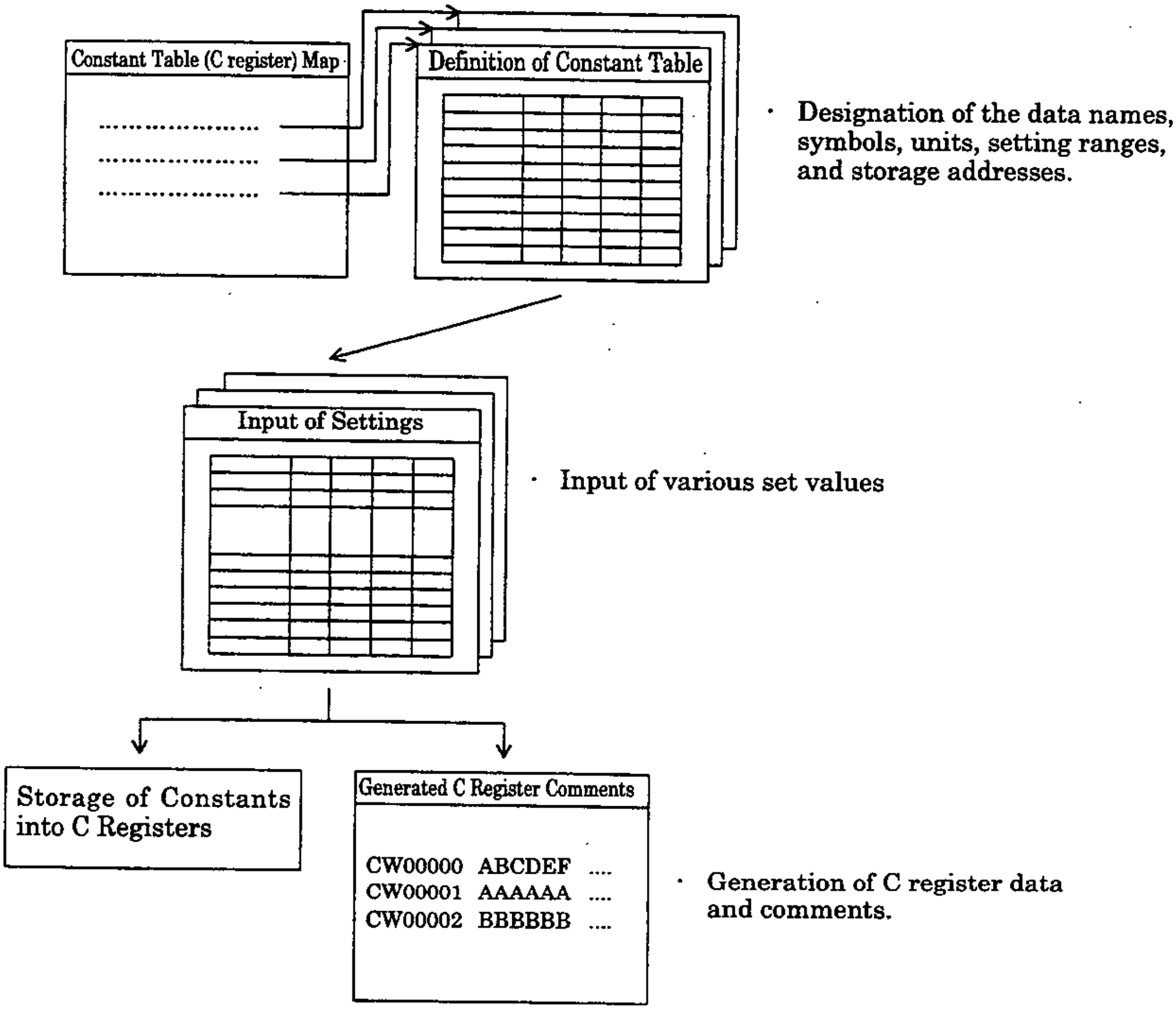

Constant Table (C register) Map
Definition of Constant Table
· Designation of the data names, symbols, units, setting ranges, and storage addresses.
Input of Settings
· Input of various set values
Storage of Constants into C Registers
Generated C Register Comments
CW00000 ABCDEF ....
CW00001 AAAAAA ....
CW00002 BBBBBB ....
· Generation of C register data and comments.

Fig. 6.7 Preparation of the C Register Constant Table

6.8.2 Preparing the Constant Table (C Register)

(1) Defining the Constant Table (C Register)

The following items should be set in defining the C register constant table. A maximum of 1638 constants may be set per page.

① **Data Name**
Designate the data name of the constant.

② **Symbol**
Designate the symbol of the constant.

③ **Unit**
Designate the unit of the constant.

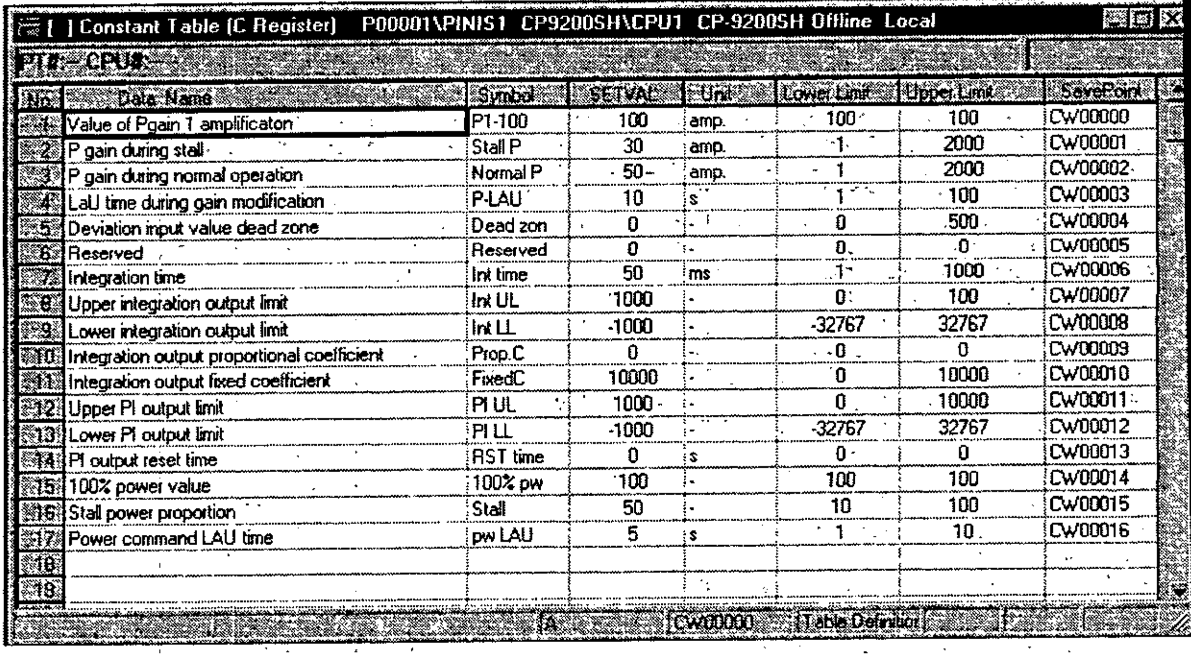
④ **Lower Limit**
Designate the lower input limit of the constant.

⑤ **Upper Limit**
Designate the upper input limit of the constant.

⑥ **Save Point**
Designate the C register into which the set values are to be stored.

(2) Inputs into the Constant Table (C Register)

The set values should be input after the definition of the C register constant table has been completed.



[ ] Constant Table (C Register) P00001\PINIS1 CP9200SH\CPU1 CP-9200SH Offline Local

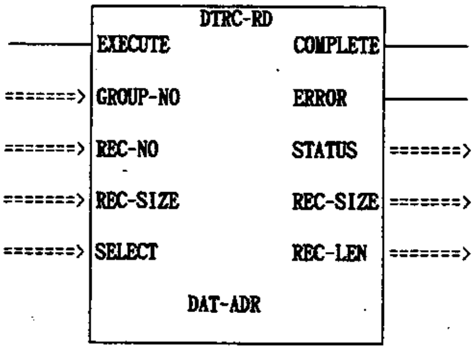
PT#– CPU#–

No.	Data Name	Symbol	SETVAL	Unit	Lower Limit	Upper Limit	SavePoint
1	Value of Pgain 1 amplificaton	P1-100	100	amp.	100	100	CW00000
2	P gain during stall	Stall P	30	amp.	1	2000	CW00001
3	P gain during normal operation	Normal P	50	amp.	1	2000	CW00002
4	LaU time during gain modification	P-LAU	10	s	1	100	CW00003
5	Deviation input value dead zone	Dead zon	0	-	0	500	CW00004
6	Reserved	Reserved	0	-	0	0	CW00005
7	Integration time	Int time	50	ms	1	1000	CW00006
8	Upper integration output limit	Int UL	1000	-	0	100	CW00007
9	Lower integration output limit	Int LL	-1000	-	-32767	32767	CW00008
10	Integration output proportional coefficient	Prop.C	0	-	0	0	CW00009
11	Integration output fixed coefficient	FixedC	10000	-	0	10000	CW00010
12	Upper PI output limit	PI UL	1000	-	0	10000	CW00011
13	Lower PI output limit	PI LL	-1000	-	-32767	32767	CW00012
14	PI output reset time	RST time	0	s	0	0	CW00013
15	100% power value	100% pw	100	-	100	100	CW00014
16	Stall power proportion	Stall	50	-	10	100	CW00015
17	Power command LAU time	pw LAU	5	s	1	10	CW00016
18							
19							

7 STANDARD SYSTEM FUNCTIONS

The functions that are provided as standard system functions and their I/O parameters are described in this chapter.

7.1 Data Trace Read Function (DTRC-RD)

Name of Function	DTRC-RD
Function	Reads out the trace data of the main controller unit and stores this data in the user registers. The data in the trace memory can be read out upon designating the record number and the number of records. The readout can be performed by designating just the necessary items in the record.
Function Definition	 DTRC-RD EXECUTE COMPLETE GROUP-NO ERROR REC-NO STATUS REC-SIZE REC-SIZE SELECT REC-LEN DAT-ADR

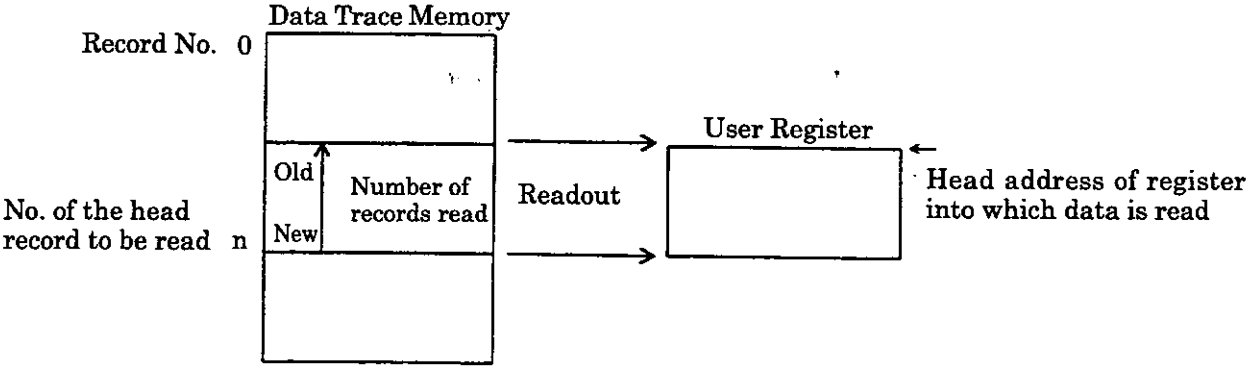
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Designation of the execution of data trace read
	2	GROUP-NO	I-REG	Designation of the data trace group No. (1 to 4)
	3	REC-NO	I-REG	Designation of the head record No. for readout (0 to maximum record number -1)
	4	REC-SIZE	I-REG	Designation of the number of records requested for readout (1 to maximum record number)
	5	SELECT	I-REG	Item to be read out (0001H to FFFFH) Bits 0 to F correspond to data designations 1 to 16 of the trace definition.
	6	DAT-ADR	Address input	Designation of the No. of the head register for readout (address of MW or DW)
Output	1	COMPLETE	B-VAL	Completion of trace read
	2	ERROR	B-VAL	Occurrence of error
	3	STATUS	I-REG	Data trace read execution status
	4	REC-SIZE	I-REG	Number of records read
	5	REC-LEN	I-REG	Length (in words) of 1 record that is read

* : Indicates the I/O designation at the CP-717.

Configuration of the Data Trace Read Execution Status (STATUS)

Name	Bit No.	Remarks
System reserved	bit0 to bit7	
No trace definition	bit8	The function will not be executed.
Group No. error	bit9	The function will not be executed.
Designated record No. error	bit10	
Error in the designated number of records read	bit11	The function will not be executed.
Data storage error	bit12	The function will not be executed.
System reserved	bit13 and bit14	
Address input error	bit15	The function will not be executed.

.1.1 Readout of Data


Data Trace Memory
Record No. 0
No. of the head record to be read n
Old
New
Number of records read
Readout
User Register
Head address of register into which data is read

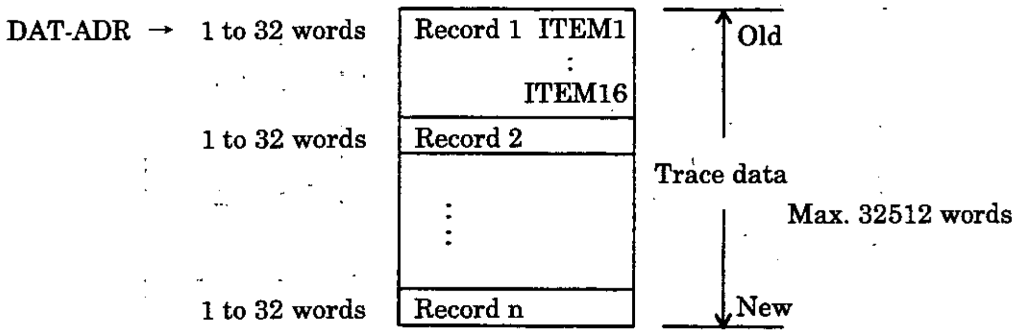
The most recent record Nos. of trace groups are each stored in SW00100 to SW00103 as shown in Table 7.1. To read the most recent trace data, designate the most recent record No. as the record No. to be read.

Table 7.1 Newest Record Number

System register number	Data trace definition
SW00100	For group 1
SW00101	For group 2
SW00102	For group 3
SW00103	For group 4
SW00104	——
SW00105	——
SW00106	——
SW00107	——

7.1.2 Configuration of the Read Data

(1) Data Configuration



(2) Record Length

A record is composed of the data for the selected items.

Word length of 1 record = Bn × 1 word + Wn × 1 word + Ln × 2 words + Fn × 2 words

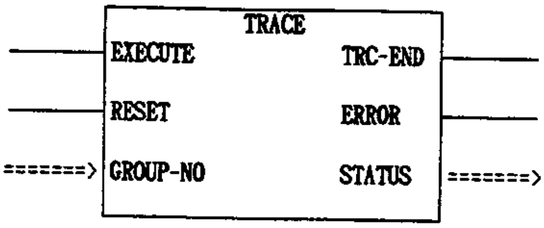
Bn: Number of bit type register selected points
Wn: Number of word type register selected points
Ln: Number of double-length integer type register selected points
Fn: Number of real number type register selected points
} A maximum of 16 points in total.

Maximum record length = 32 words (e.g. when there are 16 double-length integer type real number type registers)
Minimum record length = 1 word (e.g. when there is one bit type or integer type register)

(3) Number of Records

Maximum number of records	32512/record length
Number of records when the record length is the maximum	0 to 1016
Number of records when the record length is the minimum	0 to 32512

.2 Trace Function (TRACE)

Name of Function	TRACE
Function	Performs execution control of the tracing of the trace data designated by the trace group No. The trace is defined at "Data Trace Definition" screen (refer to the Control Pack CP-717 Operation Manual (SIE-C877-17.4, -17.5) for details). · Tracing is executed when the trace execution command (EXECUTE) is set to ON. · The trace counter is reset when the trace reset command (RESET) is set to ON. The trace end (TRC-END) output is also reset at this time. · The trace end (TRC-END) output is set to ON when the trace execution count becomes equal to the set count (set at Trace Definition).
Function Definition	 TRACE EXECUTE RESET =======> GROUP-NO TRC-END ERROR STATUS =======>

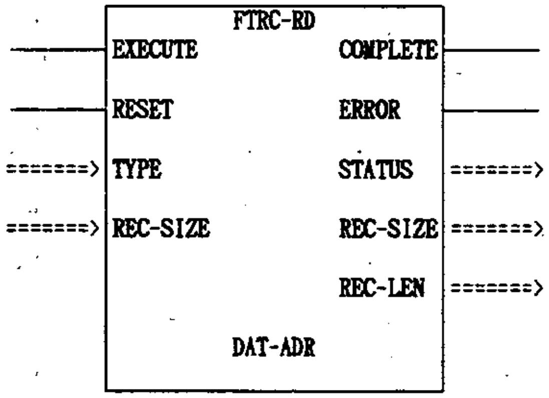
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Trace execution command
	2	RESET	B-VAL	Trace reset command
	3	GROUP-NO	I-REG	Designation of the trace group No. (1 to 4)
Output	1	TRC-END	B-VAL	End of trace
	2	ERROR	B-VAL	Occurrence of error
	3	STATUS	I-REG	Trace execution status

* : Indicates the I/O designation at the CP-717.

Configuration of the Trace Execution Status (STATUS)

Name	Bit No.	Remarks
Trace data full	bit0	This becomes ON after one round of reading of the contents in the data trace memory of the designated group has been completed.
System reserved	bit1 to bit7	
No trace definition	bit8	The function will not be executed.
Designated group No. error	bit9	The function will not be executed.
System reserved	bit10 to bit12	
Execution timing error	bit13	The function will not be executed.
System reserved	bit14	
System reserved	bit15	

7.3 Failure Trace Read Function (FTRC-RD)

Name of Function	FTRC-RD
Function	Reads the failure trace data and stores them in the user register. The data in the trac buffer can be read out upon designating the number of records needed. Either th failure occurrence data or the restoration data are designated for readout. Enables th reset (initialization) of the failure trace buffer.
Function Definition	 FTRC-RD EXECUTE COMPLETE RESET ERROR TYPE STATUS REC-SIZE REC-SIZE REC-LEN DAT-ADR

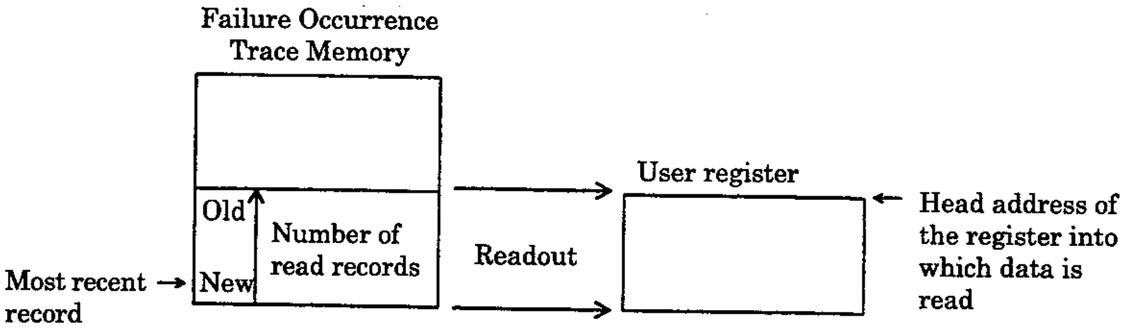
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Failure trace readout command
	2	RESET	B-VAL	Failure trace buffer reset command
	3	TYPE	I-REG	Type of data read 1 : Occurrence data 2 : Restoration data
	4	REC-SIZE	I-REG	Number of read records Occurrence data: 1 to 64 Restoration data: 450
	5	DAT-ADR	Address input	Head register address for reading (address of MW or DW)
Output	1	COMPLETE	B-VAL	Completion of failure trace read
	2	ERROR	B-VAL	Occurrence of error
	3	STATUS	I-REG	Failure trace read execution status
	4	REC-SIZE	I-REG	Number of read records
	5	REC-LEN	I-REG	Length of read record

* : Indicates the I/O designation at the CP-717.

Failure Trace Read Execution Status (STATUS)

Name	Bit No.	Remarks
System reserved	bit0 to bit7	
No trace definition	bit8	The function will not be executed.
Designated group No. error	bit9	The function will not be executed.
System reserved	bit10	
Error in the designated number of records	bit11	The function will not be executed.
Data storage error	bit12	The function will not be executed.
System reserved	bit13	
System reserved	bit14	
Address input error	bit15	The function will not be executed.

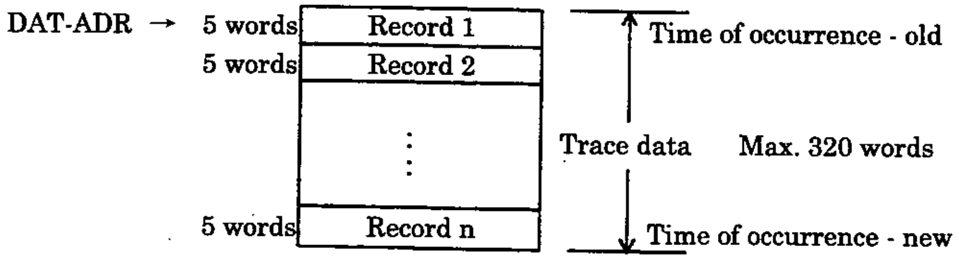
3.1 Data Readout (Failure Occurrence Data)


Failure Occurrence Trace Memory
Old
New
Number of read records
Most recent record
Readout
User register
Head address of the register into which data is read

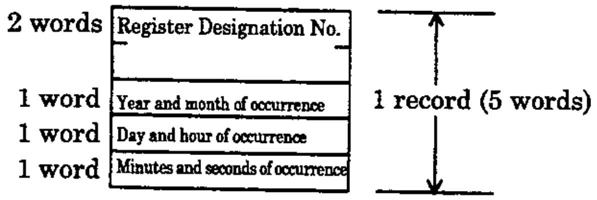
The readout will always be started from the most recent record.

3.2 Readout Data Configuration (Failure Occurrence Data)

(1) Data Configuration

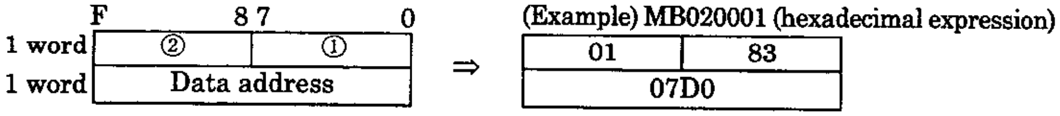


(2) Record Configuration


2 words
Register Designation No.
1 word
Year and month of occurrence
1 word
Day and hour of occurrence
1 word
Minutes and seconds of occurrence
1 record (5 words)

(3) Structure of Register Designation No. (2 words)

Contains the failure detection relay information.



	Bit Configuration of ①	Bit Configuration of ②
7	Defined flag (1 = defined, 0 = undefined)	System reserved (= 0)
6	System reserved (= 0)	Data type Bit = 0, Integer = 1, Double-length integer = 2, Real Number = 3
5		
4	0 = NO contact designation, 1 = NC contact designation	

3	Type of variable
	S=0, I=1
	O=2, M=3
		Bit address 0 to F

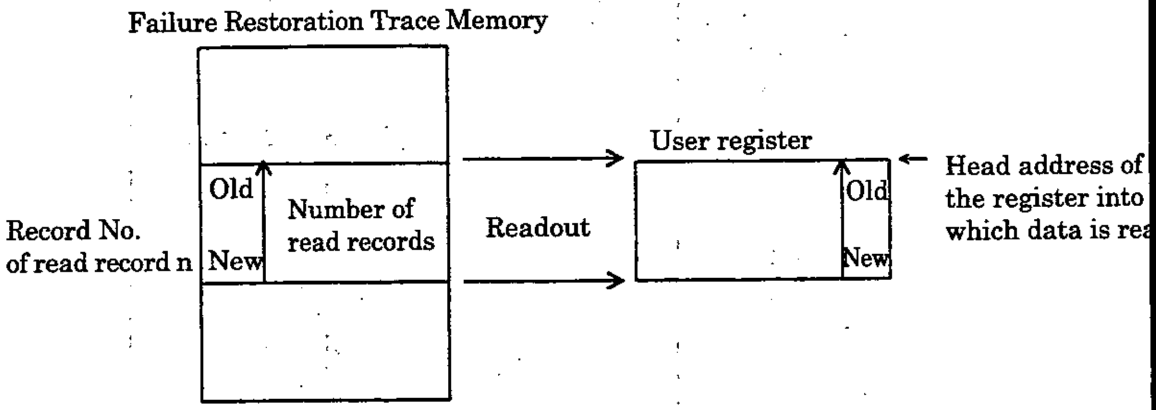
2		
1		
0		

(4) Number of Records

Minimum number of records	0
Maximum number of records	64

← 0 = no failure occurrence data

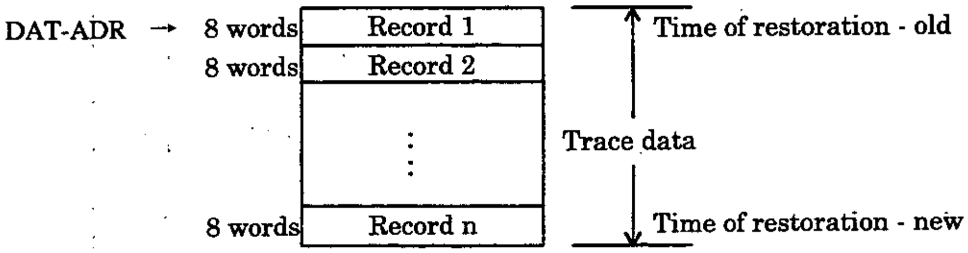
7.3.3 Data Readout (Failure Restoration Data)


Failure Restoration Trace Memory
Old
New
Number of read records
Record No. of read record n
Readout
User register
Old
New
Head address of the register into which data is rea

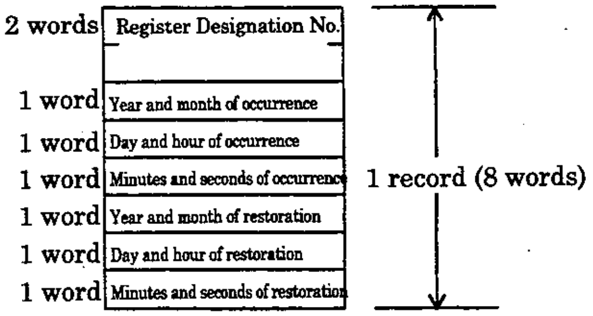
The number (amount) of restoration data is stored in SW00093 (ring counter for 1 to 9999).

7.3.4 Readout Data Configuration (Failure Restoration Data)

(1) Data Configuration



(2) Record Configuration

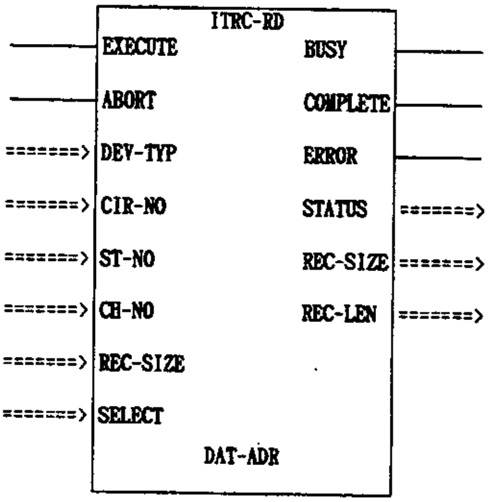

2 words
Register Designation No.
1 word
Year and month of occurrence
1 word
Day and hour of occurrence
1 word
Minutes and seconds of occurrence
1 word
Year and month of restoration
1 word
Day and hour of restoration
1 word
Minutes and seconds of restoration
1 record (8 words)

(3) Number of Records

Minimum number of records	0
Maximum number of records	450

←0 = no failure restoration da

4 Inverter Trace Read Function (ITRC-RD)

Name of Function	ITRC-RD			
Function	Reads out the trace data of the inverter and stores this data in the user registers. The data in the trace buffer can be read out upon designating the number of records needed. The readout can be performed upon designating just the necessary items in the record. [Applicable inverters] Inverters connected via CP-213, CP-215, or CP-216			
Function Definition				
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Inverter trace read command
	2	ABORT	B-VAL	Inverter trace read forced interruption command

	3	DEV-TYP	I-REG	Type of transmission device
				CP-213=2
				CP-215=1
				CP-216=4

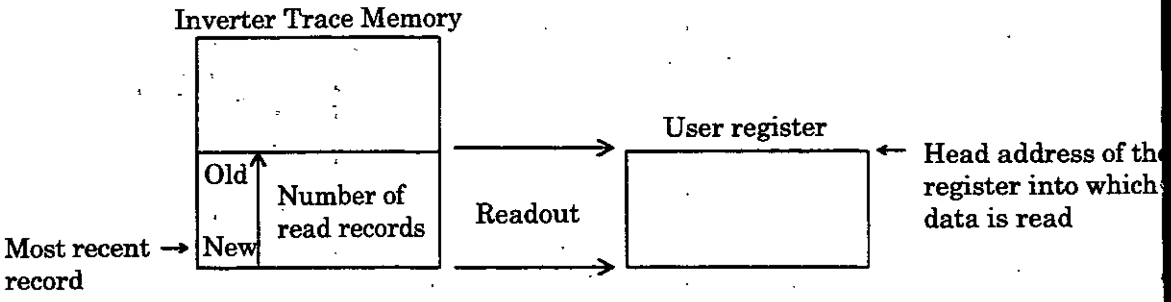
	4	CIR-NO	I-REG	Line No. CP-213: 1 to 8 CP-215: 1 to 8 CP-213: 1 to 8
	5	ST-NO	I-REG	Slave station No. CP-213: 1 to 31 CP-215: 1 to 64 CP-216: 1 to 30
	6	CH-NO	I-REG	Transmission buffer channel No. (No designation)
	7	REC-SIZE	I-REG	Number of records to be read (1 to 64)
	8	SELECT	I-REG	Items to be read (0001H to FFFFH) Bits 0 to F correspond to trace data items 1 to 16.
	9	DAT-ADR	Address input	Head address of data buffer register (address of MW or DW)
Output	1	BUSY	B-VAL	The reading of inverter trace data is in progress.
	2	COMPLETE	B-VAL	Completion of inverter trace read
	3	ERROR	B-VAL	Occurrence of error
	4	STATUS	I-REG	Inverter trace read execution status
	5	REC-SIZE	I-REG	Number of read records
	6	REC-LEN	I-REG	Length of read record (for 1 record)

* : Indicates the I/O designations at the CP-717.

Configuration of the Inverter Trace Read Execution Status (STATUS)

Name	Bit No.	Remarks
System reserved	bit0 to bit8	
Transmission parameter error	bit 9	The function is not executed.
System reserved	bit10	
Error in the designated number of records	bit11	The function is not executed.
Data storage error	bit12	The function is not executed.
Transmission error	bit13	The function is not executed.
System reserved	bit14	
Address input error	bit15	The function is not executed.

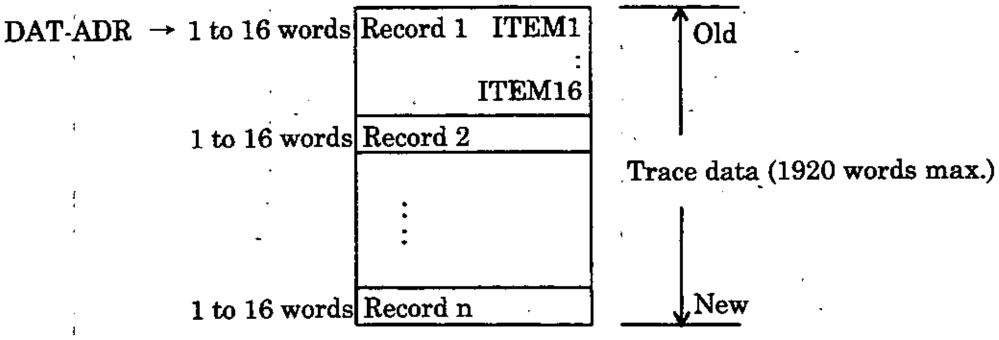
7.4.1 Readout of Inverter Trace Data



The readout will always be started from the most recent record.

7.4.2 Readout Data Configuration

(1) Data Configuration



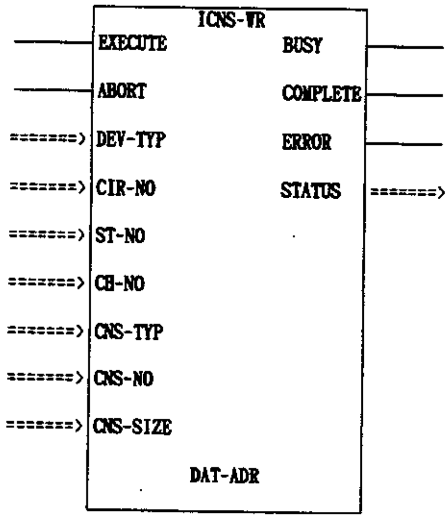
(2) Record Length

A record is composed of the data of the selected items.
Word length of 1 record = 1 to 16 words

(3) Number of Records

Maximum number of records = 120

.5 Inverter Constant Write Function (ICNS-WR)

Name of Function	ICNS-WR
Function	Writes the inverter constants. The types and ranges of the inverter constants to be written can be designated. [Applicable inverters] Inverters connected via CP-215, or CP-216
Function Definition	 ICNS-WR EXECUTE ABORT DEV-TYP CIR-NO ST-NO CH-NO CNS-TYP CNS-NO CNS-SIZE DAT-ADR BUSY COMPLETE ERROR STATUS

I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Inverter constant write command
	2	ABORT	B-VAL	Inverter constant write forced interruption command

	3	DEV-TYP	I-REG	Type of transmission device
				CP-215=1
				CP-216=4

	4	CIR-NO	I-REG	Line No. CP-215: 1 to 8 CP-216: 1 to 8
	5	ST-NO	I-REG	Slave station No. CP-215: 1 to 64 CP-216: 1 to 30
	6	CH-NO	I-REG	Transmission buffer channel No. (No designation)
	7	CNS-TYP	I-REG	Type of inverter constant 0 = direct designation of reference No., 1 = An, 2 = Bn, 3 = Cn, 4 = Dn, 5 = En, 6 = Fn, 7 = Hn, 8 = Ln, 9 = On, 10=Tn
	8	CNS-NO	I-REG	Inverter constant No. (1 to 99) The upper limit will differ according to the type of inverter. If CNS-TYP = 0, designate the reference No.
	9	CNS-SIZE	I-REG	Number of inverter constants (number of data to be written) 1 to 100
	10	DAT-ADR	Address input	Register address of set data (address of MW, DW, or #W)
Output	1	BUSY	B-VAL	Inverter constants are being written in.
	2	COMPLETE	B-VAL	The write-in of inverter constants has been completed.
	3	ERROR	B-VAL	Occurrence of error
	4	STATUS	I-REG	Inverter constant write execution status

* : Indicates the I/O designations at the CP-717.

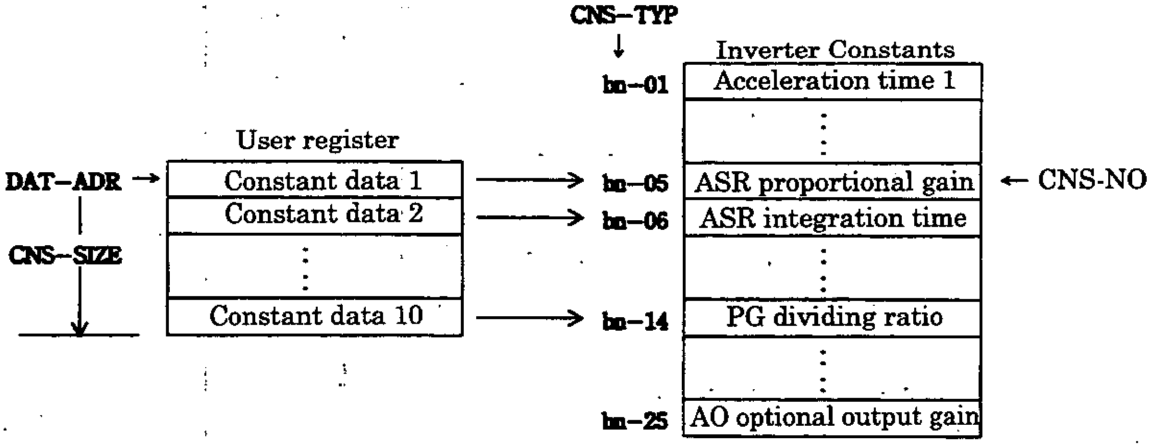
Configuration of Inverter Constant Write Execution Status (STATUS)

Name	Bit No.	Remarks
System reserved	bit0 to bit7	
Execution sequence error	bit 8	The function is not executed.
Transmission parameter error	bit 9	The function is not executed.
Designated type error	bit10	The function is not executed.
Designated No. error	bit11	The function is not executed.
Error in number (amount) of the designated data	bit12	The function is not executed.
Transmission error	bit13	The function is not executed.
Inverter response error	bit14	The function is not executed.
Address input error	bit15	The function is not executed.

(Note) : In the case of an inverter response error, the error codes from the inverter are indicated in bit0 to bit7.

01H(1) : function code error
02H(2) : reference No. error
03H(3) : write-in count error
21H(33) : write-in data upper/lower limit error
22H(34) : write-in error (during running, during UV)
Numbers in ( ) are of decimal expressions.

7.5.1 Configuration of the Write-in Data



5.2 Method of Writing to an EEPROM

Procedures for writing constants to an EEPROM (inverter internal constant storage memory) are shown in Fig. 7.1.

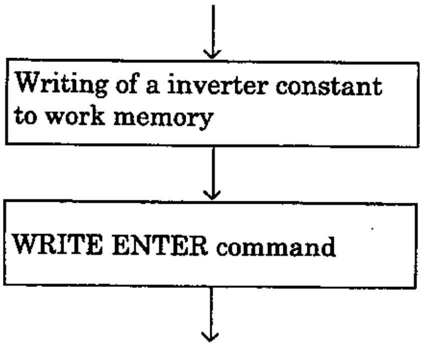


Fig. 7.1 EEPROM Write Procedures

Constants written with the system function "ICNS-WR" are once entered in work memory. In order to actually store these in EEPROM, it is necessary to bring up the WRITE ENTER command as shown in Fig. 7.2.

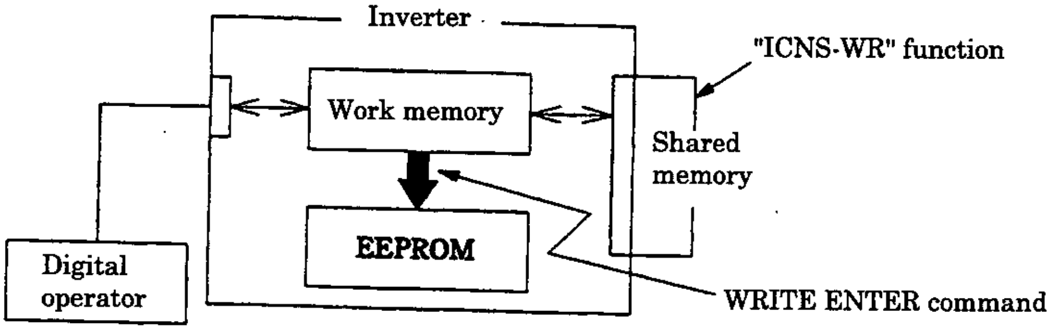


Fig. 7.2 WRITE ENTER Command

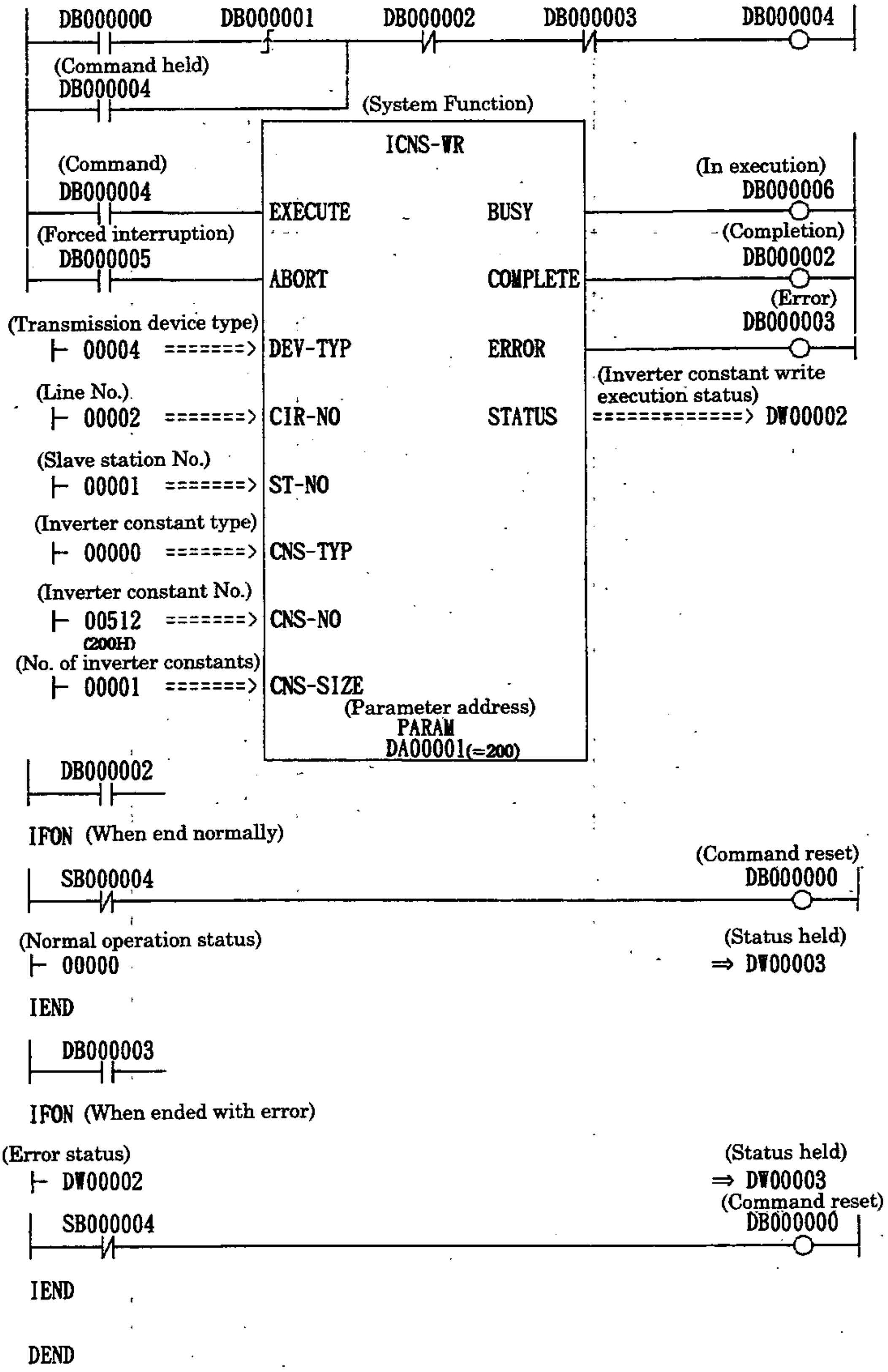
(1) WRITE ENTER Command

Using the "ICNS-WR" function, by writing the data "0" in the reference number "FFFD," the WRITE ENTER command is entered for the inverter.

(2) Program Example

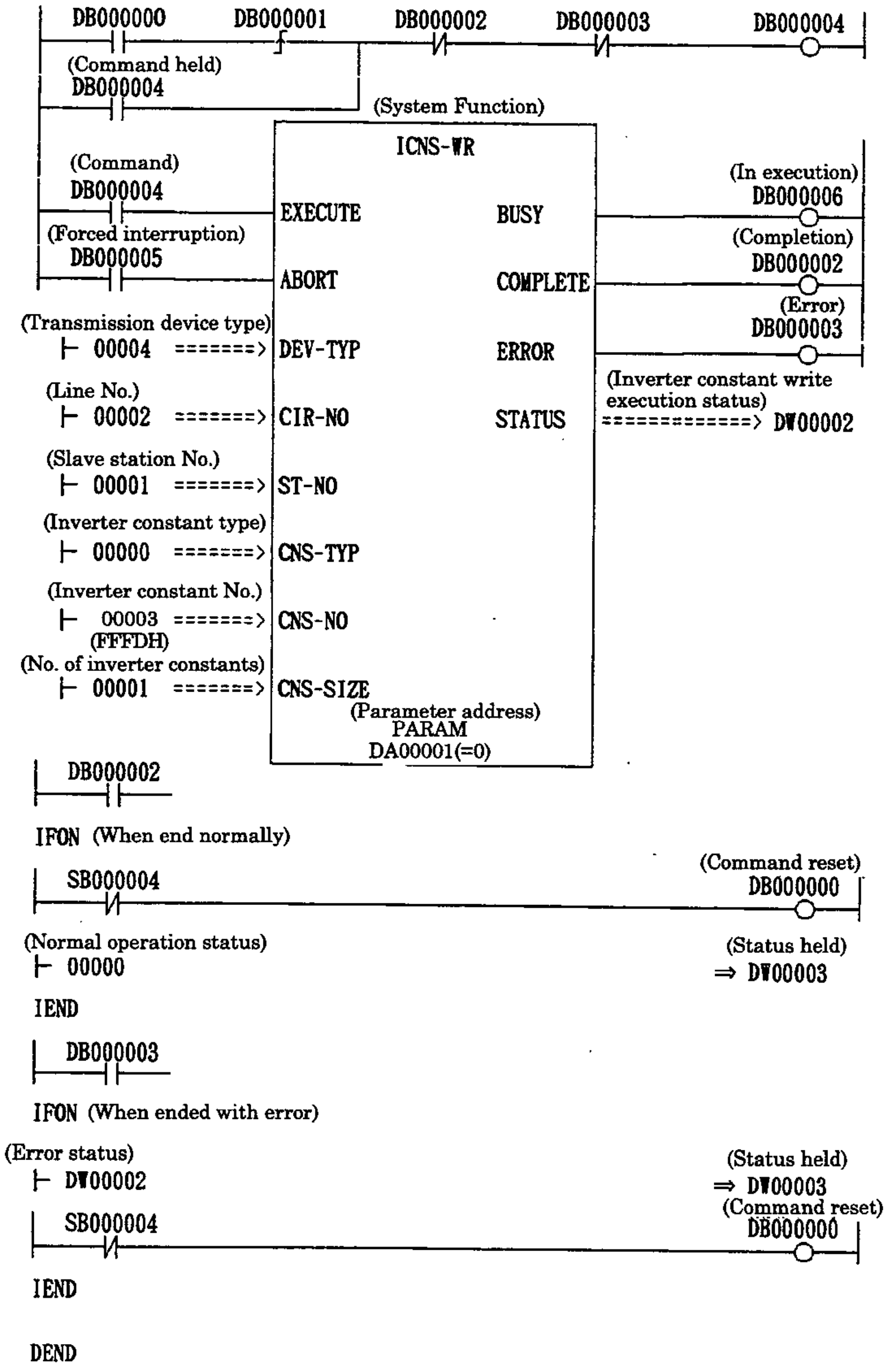
An example of a program that writes “200” in the constant “C1-01” is shown in Fig. 7.3 (①, ②

① First, write to the inverter work memory.



*: By turning DB00000 = ON, a one time only write can be executed.

② Actually writing to EEPROM. (Enter the WRITE ENTER command.)

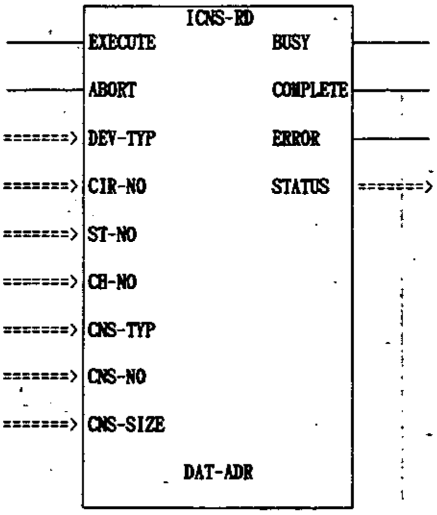


*: By turning DB00000 = ON, a one time only write can be executed.

Fig. 7.3 Program Example

> **NOTE**
> - The WRITE ENTER command writes all constants that have been written to work memory up to that point to the EEPROM.
> - If power to the inverter is turned OFF, work memory data is lost, but data written to the EEPROM is saved.

7.6 Inverter Constant Read Function (ICNS-RD)

Name of Function	ICNS-RD
Function	Reads the inverter constants. The types and ranges of the inverter constants to be read can be designated. [Applicable inverters] Inverters connected via CP-213, CP-215, or CP-216.
Function Definition	 ICNS-RD EXECUTE ABORT DEV-TYP CIR-NO ST-NO CH-NO CNS-TYP CNS-NO CNS-SIZE DAT-ADR BUSY COMPLETE ERROR STATUS

I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Inverter constant read execution command
	2	ABORT	B-VAL	Inverter constant read forced interruption command

	3	DEV-TYP	I-REG	Type of transmission device
				CP-215=1
				CP-216=4

	4	CIR-NO	I-REG	Line No. CP-215: 1 to 8 CP-216: 1 to 8
	5	ST-NO	I-REG	Slave station No. CP-215: 1 to 64 CP-216: 1 to 30
	6	CH-NO	I-REG	Transmission buffer channel No. (No designation)
	7	CNS-TYP	I-REG	Type of inverter constant 0 = direct designation of reference No. 1 = An, 2 = Bn, 3 = Cn, 4 = Dn, 5 = En, 6 = Fn, 7 = Hn, 8 = Ln, 9 = On, 10 = Tn
	8	CNS-NO	I-REG	Inverter constant No. (1 to 99) The upper limit will differ according to the type of inverter. If CNS-TYP = 0, designate the reference No.
	9	CNS-SIZE	I-REG	Number of inverter constants (number of data to be read) 1 to 125
	10	DAT-ADR	Address input	Register address of the data to be read (address of MW or DW)
Output	1	BUSY	B-VAL	Inverter constants are being read.
	2	COMPLETE	B-VAL	The reading of inverter constants has been completed.
	3	ERROR	B-VAL	Occurrence of error
	4	STATUS	I-REG	Inverter constant read execution status

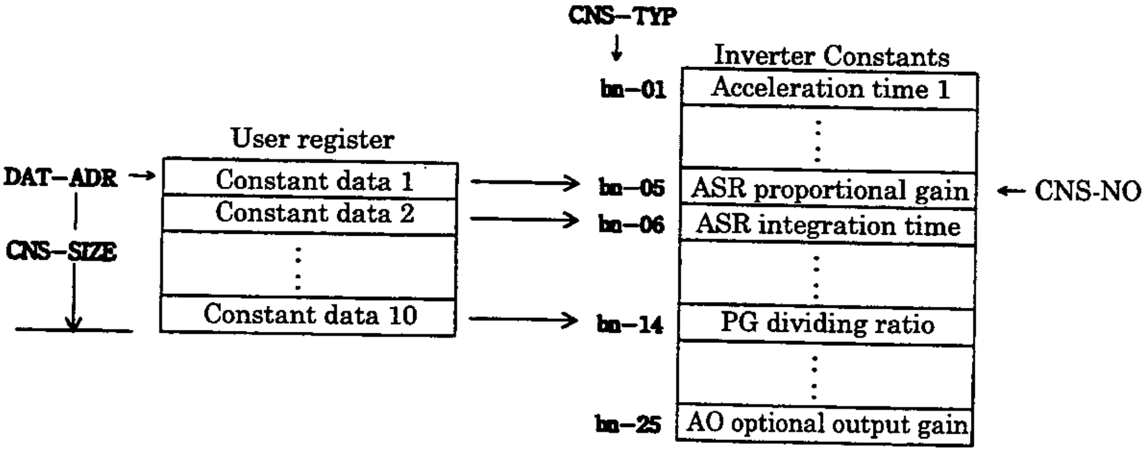
* : Indicates the I/O designations at the CP-717.

Configuration of Inverter Constant Read Execution Status (STATUS)

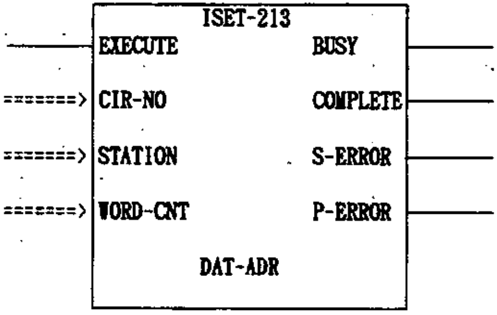
Name	Bit No.	Remarks
System reserved	bit0 to bit7	
Execution sequence error	bit 8	The function is not executed.
Transmission parameter error	bit 9	The function is not executed.
Designated type error	bit10	The function is not executed.
Designated No. error	bit11	The function is not executed.
Error in number (amount) of the designated data	bit12	The function is not executed.
Transmission error	bit13	The function is not executed.
Inverter response error	bit14	The function is not executed.
Address input error	bit15	The function is not executed.

(Note) : In the case of an inverter response error, the error codes from the inverter are indicated in bit0 to bit7.
01H(1) : function code error
02H(2) : reference No. error
03H(3) : Readout count error
Numbers in ( ) are of decimal expressions.

Configuration of the Data Readout

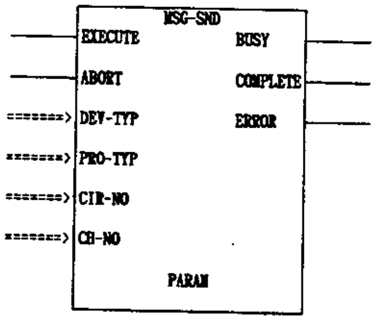


7.7 CP-213 Initial Data Setting Function (ISET-213)

Name of Function	ISET-213			
Function	Sets the initial data for the inverter connected to the CP-213 line. A few scans are required until the completion of the process.			
Function Definition	 ISET-213 EXECUTE CIR-NO STATION WORD-CNT DAT-ADR BUSY COMPLETE S-ERROR P-ERROR			
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	CP-213 initial data setting command
	2	CIR-NO	I-REG	CP-213 line No. (1 to 8)
	3	STATION	I-REG	Slave station No. (1 to 31)
	4	WORD-CNT	I-REG	Number of words of set data (1 to 127)
	5	DAT-ADR	Address input	Head address of set data (MW, DW, #W)
Output	1	BUSY	B-VAL	CP-213 initial data setting in process
	2	COMPLETE	B-VAL	Completion of CP-213 initial data setting
	3	S-ERROR	B-VAL	Occurrence of error
	4	P-ERROR	B-VAL	Parameter error

* : Indicates the I/O designation at the CP-717.

.8 Send Message Function (MSG-SND)

Name of Function	MSG-SND			
Function	Sends a message to the called station which is on the line and which is designated by the transmission device type. Supports a plurality of protocol types. The execution command (EXECUTE) must be held until COMPLETE or ERROR becomes ON. [Transmission Devices] CP-215, CP-216, CP-217, CP-218, CP-2500, CP-2520 [Protocols] MEMOBUS, non-procedural, MELSEC, OMRON			
Function Definition	 MSG-SND EXECUTE ABORT DEV-TYP PRO-TYP CIR-NO CH-NO PARAM BUSY COMPLETE ERROR			
I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Send message command
	2	ABORT	B-VAL	Send message forced interruption command

	3	DEV-TYP	I-REG	Type of transmission device
				CP-215 = 1
				CP-216 = 4
				CP-217 = 5
				CP-218 = 6
				CP-2500 = 3
				CP-2520 = 7

	4	PRO-TYP	I-REG	Transmission protocol ** MEMOBUS = 1 non-procedural = 2

	5	CIR-NO	I-REG	Line No.
				CP-215 = 1 to 8
				CP-216 = 1 to 8
				CP-217 = 1 to 24
				CP-218 = 1 to 8
				CP-2500 = 1 to 8
				CP-2520 = 1 to 8

	6	CH-NO	I-REG	Transmission buffer channel No.
				CP-215 = 1 to 13
				CP-216 = 1 to 3
				CP-217 = 1
				CP-218 = 1 to 10
				CP-2500 = 1 to 14
				CP-2520 = 1 to 15

	7	PARAM	Address input	Head address of set data (MW, DW, #W)
Output	1	BUSY	B-VAL	Message is being sent.
	2	COMPLETE	B-VAL	The sending of the message has been completed.
	3	ERROR	B-VAL	Occurrence of error

* : Indicates the I/O designation at the CP-717.

** : Designate the MEMOBUS protocol (= 1) if transmission is to be performed with the MELSEC or OMRON procedure. Protocol conversion will be carried out at the transmission device (CP-217, CP-218). Refer to (1) of 5.3.4, "OMRON Communication" or (2) of 5.3.4, "MELSEC Communication" of the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details on the protocol conversion specifications.

PARAM

No.	IN/OUT	Contents		Remarks
		MEMOBUS	Non-procedural	
00	OUT	Process result	Process result	
01	OUT	Status	Status	
02	IN	Called station #	Called station #	Called connection # in the case of DEV-TYP = CP-218
03	SYS	System reserved	System reserved	
04	IN	Function code		
05	IN	Data address	Data address	
06	IN	Data size	Data size	
07	IN	Called CPU #	Called CPU #	
08	IN	Coil offset		
09	IN	Input relay offset		
10	IN	Input register offset		
11	IN	Holding register offset		
12	SYS	For system use	For system use	
13	SYS	System reserved	System reserved	
14	SYS	System reserved	System reserved	
15	SYS	System reserved	System reserved	
16	SYS	System reserved	System reserved	

7.8.1 Parameters

(1) Process Result (PARAM00)

The process result is output to the upper byte. The lower byte is for system analysis.

- 00□□:In process (BUSY)
- 10□□:End of process (COMPLETE)
- 8□□□:Occurrence of error (ERROR)

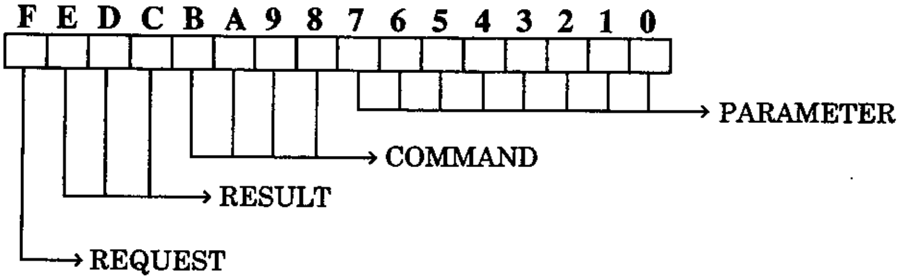
[Error Classification]

- 81□□: Function code error
  The sending of an unused function code was attempted. Or, an unused function code was received.
- 82□□: Address setting error
  The data address, coil offset, input relay offset, input register offset, or holding register offset setting is out of range.
- 83□□: Data size error
  The size of the sent or received data is out of range.
- 84□□: Line No. setting error
  The line No. setting is out of range.
- 85□□: Channel No. setting error
  The channel No. setting is out of range.
- 86□□: Station address error
  The station No. setting is out of range.
- 88□□: Transmission unit error.
  An error response was returned from the transmission unit. (Refer to (2) of 7.8.1.)
- 89□□: Device selection error
  A non-applicable device is selected.

(2) Status (PARM01)

Outputs the status of the transmission unit.

(a) Bit Assignment



(b) COMMAND

Code	Symbol	Meaning
1	U_SEND	Send generic message.
2	U_REC	Receive generic message.
3	ABORT	Forced interruption
8	M_SEND	Send MEMOBUS command ··· completed upon receipt of response.
9	M_REC	Receive MEMOBUS command ··· accompanies sending of response.
C	MR_SEND	Send MEMOBUS response.

(c) RESULT

Code	Symbol	Meaning
1	SEND_OK	Sending has been completed correctly.
2	REC_OK	Receiving has been completed correctly.
3	ABORT_OK	Completion of forced interruption
4	FMT_NG	Parameter format error
5	SEQ_NG, or INIT_NG	Command sequence error The token has not been received yet. Not connected to a transmission system.
6	RESET_NG, or O_RING_NG	Reset state Out-of-ring. The token could not be received even when the token monitor time was exceeded.
7	REC_NG	Data receive error (error detected by a program of a lower rank)

(d) PARAMETER

One of the error codes of Table 7.2 is indicated if RESULT = 4(FMT_NG). Otherwise, this indicates the address of the called station.

Table 7.2 Error Codes

Code	Error
00	No errors.
01	Station address is out of range.
02	Monitored MEMOBUS response receiving time error
03	Resending count setting error
04	Cyclic area setting error
05	Message signal CPU No. error
06	Message signal register No. error
07	Message signal word count error

(e) REQUEST

1 = Request
0 = Completion of receipt report

(3) Called Station # (PARAM02)

[CP-215]
1 to 64 : Message is sent to the designated station.
00FFH : Message is sent to all stations (broadcasting).
[CP-216]
1 to 30 : Message is sent to the designated station (possible only sending from the master statio
80H : Message is sent to the master station (possible only sending from a slave station).
Note : With CP-216, message transmission between slave stations is not possible.
[CP-217]
1 to 254: Message is sent to the station of designated device address.
[CP-218]
1 to 20 : Message is sent to the station of designated connection No.
[CP-2500]
1 to 32 : Message is sent to the designated station.
129 to 160 : Message is sent to the stations of designated group address (group transmissio
00FFH: Message is sent to all stations (broadcasting).
[CP-2520]
1 to 64 : Message is sent to the designated station.
00FFH : Message is sent to all stations (broadcasting).

(4) Function Code (PARAM04)

The MEMOBUS function code to be sent is set.

Function code		Setting
00H	Unused	×
01H	Read coil status	○
02H	Read input relay status	○
03H	Read contents of holding register	○
04H	Read contents of input register	○
05H	Change status of single coil	○
06H	Write into a single holding register	○
07H	Unused	×
08H	Loop-back test	○
09H	Read contents of holding register (expanded)	○
0AH	Read contents of input register (expanded)	○
0BH	Write into holding register (expanded)	○
0CH	Unused	×
0DH	Discontinuous readout of holding register (expanded)	○
0EH	Discontinuous write into holding register (expanded)	○
0FH	Change status of a multiple coil	○
10H	Write into a plurality of holding registers	○
11H to 20H	Unused	×
21H to 3FH	System reserved	×
40H to 4FH	System reserved	×
50H or more	Unused	×

(× : cannot be set, ○ : can be set)

Note : Only MW (MB) can be used as the sending/receiving register during master operation. T
MB, MW, IB, and IW registers can be used respectively as the coil, holding register, inp
relay, and input registers during slave operation.

(5) Data Address (PARAM05)

The set contents will differ according to the function code as follows.

① Request for readout from/write-in to coil or relay: Set the head bit address of the data.
② Request for continuous readout from/write-in to a register: Set head word address of the data.
③ Request for discontinuous readout from/write-in to a register: Set head word address of the address table.

Function code		Data Address Setting Range		
00H	Unused	Invalid		
01H	Read coil status	0 to 65535	(0 to FFFFH)	①
02H	Read input relay status	0 to 65535	(0 to FFFFH)	①
03H	Read contents of hold register	0 to 32767	(0 to 7FFFH)	②
04H	Read contents of input register	0 to 32767	(0 to 7FFFH)	②
05H	Change status of single coil	0 to 65535	(0 to FFFFH)	①
06H	Write into a single holding register	0 to 32767	(0 to 7FFFH)	②
07H	Unused	Invalid		
08H	Loop-back test	Invalid		
09H	Read contents of holding register (expanded)	0 to 32767	(0 to 7FFFH)	②
0AH	Read contents of input register (expanded)	0 to 32767	(0 to 7FFFH)	②
0BH	Write into holding register (expanded)	0 to 32767	(0 to 7FFFH)	②
0CH	Unused	Invalid		
0DH	Discontinuous readout of holding register (expanded)	0 to 32767	(0 to 7FFFH)	③
0EH	Discontinuous write into holding register (expanded)	0 to 32767	(0 to 7FFFH)	③
0FH	Change status of a multiple coil	0 to 65535	(0 to FFFFH)	①
10H	Write into a plurality of holding registers	0 to 32767	(0 to 7FFFH)	②

(6) Data Size (PARAM06)

Set the size (in number of bits or number of words) of the data that is requested for readout or write-in. The setting range will differ according to the transmission module and the function code to be used.

[CP-215]

Function code		Data Size Setting Range	
		CP-215/CP-218/CP-2520	CP-216/CP-217-CP-2500
00H	Unused	Invalid	
01H	Read coil status	1 to 2000 (1 to 07D0H)/number of bits	
02H	Read input relay status	1 to 2000 (1 to 07D0H)/number of bits	
03H	Read contents of holding register	1 to 125 (1 to 007DH)/number of words	
04H	Read contents of input register	1 to 125 (1 to 007DH)/number of words	
05H	Change status of single coil	Invalid	
06H	Write into a single holding register	Invalid	
07H	Unused	Invalid	
08H	Loop-back test	Invalid	
09H	Read contents of holding register (expanded)	1 to 508 (1 to 01FCH)/number of words	1 to 252 (1 to 00FCH)/number of words
0A	Read contents of input register (expanded)	1 to 508 (1 to 01FCH)/number of words	1 to 252 (1 to 00FCH)/number of words
0B	Write into holding register (expanded)	1 to 507 (1 to 01FBH)/number of words	1 to 251 (1 to 00FBH)/number of words
0C	Unused	Invalid	
0D	Discontinuous readout of holding register (extended)	1 to 508 (1 to 01FCH)/number of words	1 to 252 (1 to 00FCH)/number of words
0E	Discontinuous write into holding register (extended)	1 to 254 (1 to 00FEH)/number of words	1 to 126 (1 to 007EH)/number of words
0FH	Change status of multiple coil	1 to 800 (1 to 0320H)/number of bits	
10H	Write into a plurality of holding registers	1 to 100 (1 to 0064H)/number of words	

(7) Called CPU # (PARAM07)

Set the called CPU No.
When the sending destination is CP-9200SH, set 1 or 2.
For other cases, set 0.

(8) Coil Offset (PARAM08)

Set the offset word address of the coil.
This is valid in the case of function codes 01H, 05H, and 0FH.

(9) Input Relay Offset (PARAM09)

Set the offset word address of the input relay.
This is valid in the case of function code 02H.

(10) Input Register Offset (PARAM10)

Set the offset word address of the input register.
This is valid in the case of function codes 04H and 0AH.

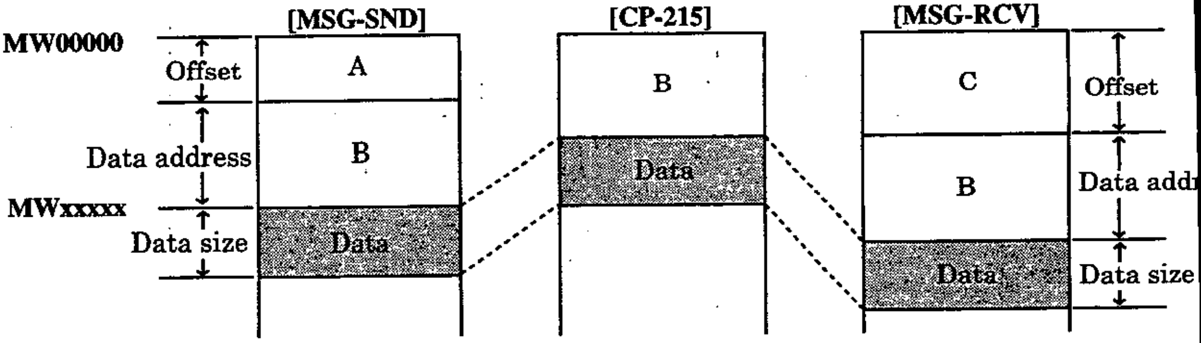
(11) Holding Register Offset (PARAM11)

Set the offset word address of the holding register.
This is valid in the case of function codes 03H, 06H, 09H, 0BH, 0DH, 0EH, and 10H.

(12) For System Use (PARAM12)

The channel No. being used is stored. Make sure that this will be set to 0000H by the us program on the first scan after turning on the power. This parameter must not be changed by th user program thereafter since this parameter will then be used by the system.

(13) Relationship between the Data Address, Size and Offset



A = sending side offset address
B = sending side data address
C = receiving side offset address

(14) When transmission protocol is set to non-procedural

The settings of PARAM04, PARAM08, PARAM09, and PARAM10 are not necessary. Transmissio enabled register is only MW.

.8.2 Inputs

(1) EXECUTE (Send Message Execution Command)

When this command becomes "ON", the message is sent.

This must be held until COMPLETE (completion of process) or ERROR (occurrence of error) becomes "ON".

(2) ABORT (Send Message Forced Interruption Command)

This command forcibly interrupts the sending of the message. This has priority over EXECUTE (send message execution command).

(3) DEV-TYP (Transmission Device Type)

Designates transmission device type.

	Transmission Device Type
CP-215	1
CP-216	4
CP-217	5
CP-218	6
CP-2500	3
CP-2520	7

(4) PRO-TYP (Transmission Protocol)

Designates transmission protocol. When transmitting with MELSEC or OMRON procedures, specify MEMOBUS protocol (=1). Protocol is converted by the transmission device (CP-217, CP-218).

MEMOBUS: Setting = 1
Non-procedural: Setting = 2

For details of protocol conversion specifications, refer to the following manuals.

Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (1) "OMRON communications"
5.3.4 (2) "MELSEC communications"

Note: In non-procedural transmission, a response is not received from the other station.

(5) CIR-NO (Circuit No.)

Designate the Circuit No.

	Circuit No.
CP-215	1 to 8 (Option)
CP-216	1 to 8 (Option)
CP-217	1 to 24 (Option)
CP-218	1 to 8 (Option)
CP-2500	1 to 8 (Option)
CP-2520	1 to 8 (Option)

(6) CH-NO (Channel No.)

Designate the channel No. of the transmission unit. However, the channel number should be set so as not to be duplicated on a single line.

	Channel No.
CP-215	1 to 13
CP-216	1 to 3
CP-217	1
CP-218	1 to 10
CP-2500	1 to 14
CP-2520	1 to 15

(7) PARAM (Set Data Head Address)

The head address of the set data is designated. For details of the set data refer to 7.8.1. "Parameters."

7.8.3 Outputs

(1) BUSY (In Process)
Indicates that the process is being executed. Keep EXECUTE set to "ON".

(2) COMPLETE (Completion of Process)
Becomes "ON" for only 1 scan upon normal completion.

(3) ERROR (Occurrence of Error)
Becomes "ON" for only 1 scan upon occurrence of error.
Refer to PARAM00 (7.8.1 (1)) and PARAM01 (7.8.1 (2)) concerning the cause.

8.4 Limitations Arising from Other Companies' Communications Protocols with the CP-217IF

(1) When Making a Dedicated Protocol Connection Link with the MELSEC Computer

■ Communication is possible with type 1 protocol (response possible only for full-dual connection).

■ With a MSG-SND function, receiving and sending with responce of ACPU common commands to and from the MELSEC sequencer are possible, but commands that may be used are limited (read out/write in of device memory, wrap test).

■ Designate MEMOBUS protocol (=1) for input of the PRO-TYP (transmission protocol) of the MSG-SND function. On the I/O definition screen for the transmission port, if MELSEC master is set, conversion to the corresponding MELSEC format is performed by the CP-217IF unit. Change designated parameters at this time to parameters of corresponding MEMOBUS procedures.
Refer to the following manuals for correspondence of MELSEC commands and MEMOBUS function codes, and correspondence of registers for sending and receiving and device addresses on the MELSEC side.
· Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (2) "MELSEC communications"

■ In MEMOBUS → MELSEC format conversion, due to MELSEC protocol characteristic restrictions or MELSEC sequencer type characteristic restrictions, limits in addition to number of read out words of a register and other MEMOBUS procedures arise, so carefully read manuals related to connected equipment before using.
Furthermore be sure to refer to the manual related to MELSEC computer link dedicated protocol type 1 commands.

(2) When Making an OMRON Upward Linking Mode (SYSWAY) Connection

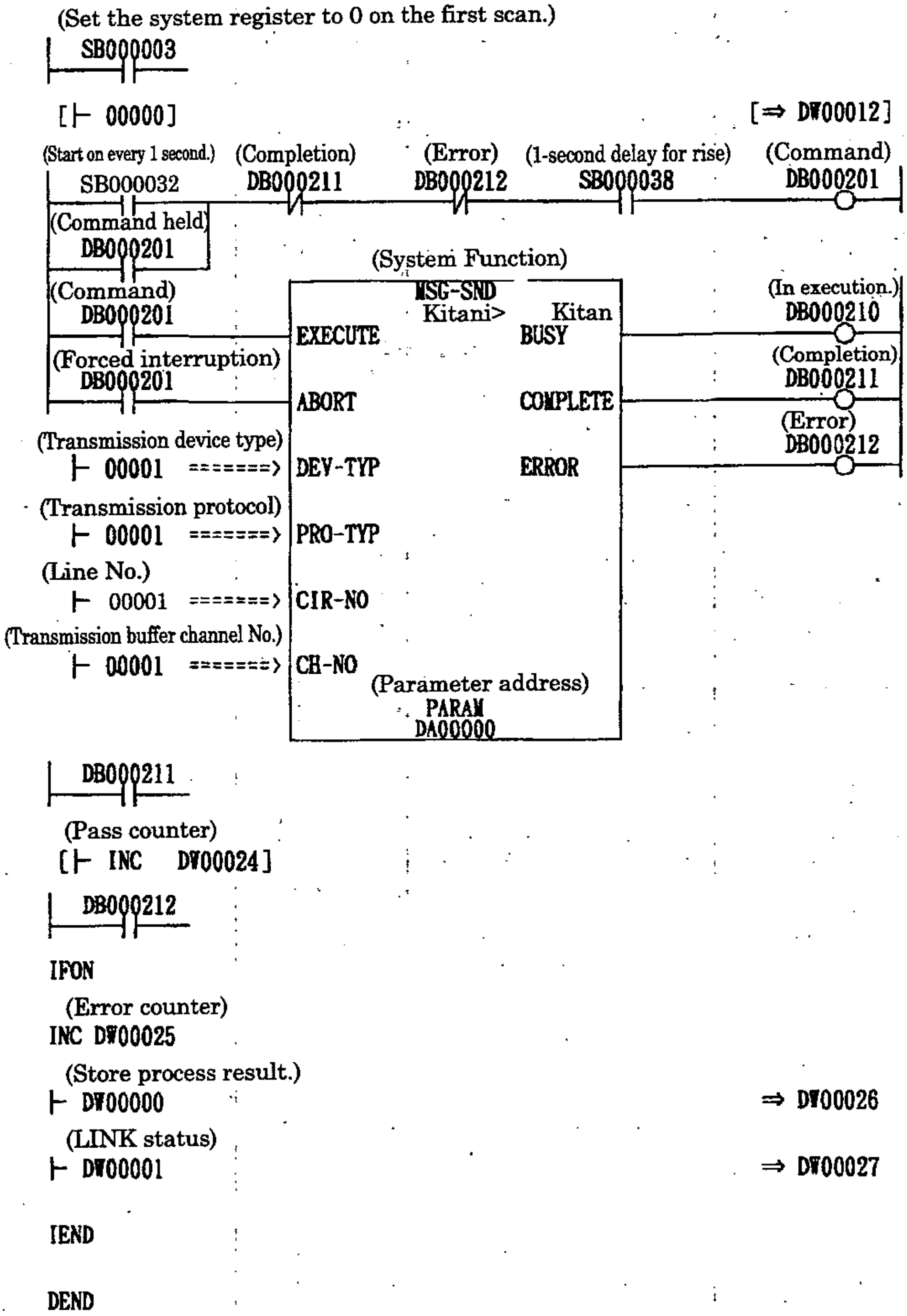
■ With a MSG-SND function, sending and receiving with response of commands to and from the OMRON sequencer are possible, but commands that may be used are limited (I/O relay/DM read out/write, wrap test).

■ Designate MEMOBUS protocol (=1) for input of the PRO-TYP (transmission protocol) of the MSG-SND function. On the I/O definition screen for the transmission port, if OMRON master is set, conversion to the corresponding OMRON format is performed by the CP-217IF unit. Change designated parameters at this time to parameters of corresponding MEMOBUS procedures.
Refer to the following manuals for correspondence of OMRON commands and MEMOBUS function codes, and regarding correspondence of registers for sending and receiving and the relay (CH)/DM area on the OMRON side.
· Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (1) "OMRON communications"

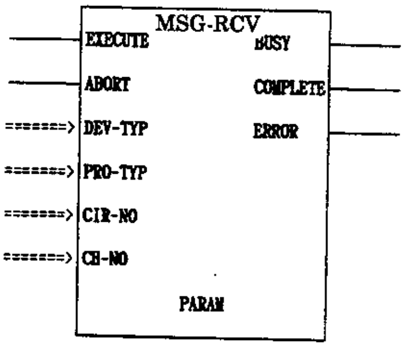
■ In MEMOBUS → OMRON format conversion, due to OMRON protocol characteristic restrictions or OMRON sequencer type characteristic restrictions, limits in addition to number of read out words of a register and other MEMOBUS procedures arise, so carefully read manuals related to connected equipment before using.
Furthermore be sure to refer to the manual related to OMRON communications procedures.

■ This corresponds to transmission procedures by multi-programs stipulated in OMRON procedures, but set the upper limit for the number of words that can be accessed with one instruction to 125 words for DM register read out, and 100 words for write-in (restricted conditions of MEMOBUS procedures).

7.8.5 Program Example



9 Receive Message Function (MSG-RCV)

Name of Function	MSG-RCV
Function	Receives a message from a calling station which is on the line and which is designated by the transmission device type. Supports a plurality of protocol types. The execution command (EXECUTE) must be held until COMPLETE or ERROR becomes ON. [Transmission Devices] CP-215, CP-216, CP-217, CP-218, CP-2500, CP-2520 [Protocols] MEMOBUS, non-procedural, MELSEC, OMRON
Function Definition	 MSG-RCV EXECUTE ABORT DEV-TYP PRO-TYP CIR-NO CH-NO PARAM BUSY COMPLETE ERROR

I/O Definition	No.	Name	I/O Designation*	Description
Input	1	EXECUTE	B-VAL	Receive message command
	2	ABORT	B-VAL	Receive message forced interruption command

	3	DEV-TYP	I-REG	Type of transmission device
				CP-215 = 1
				CP-216 = 4
				CP-217 = 5
				CP-218 = 6
				CP-2500 = 3
				CP-2520 = 7

	4	PRO-TYP	I-REG	Transmission protocol ** MEMOBUS = 1 non-procedural = 2

	5	CIR-NO	I-REG	Line No.
				CP-216 = 1 to 8
				CP-215 = 1 to 8
				CP-217 = 1 to 24
				CP-218 = 1 to 8
				CP-2500 = 1 to 8
				CP-2520 = 1 to 8

	6	CH-NO	I-REG	Transmission buffer channel No.
				CP-215 = 1 to13
				CP-216 = 1 to 3
				CP-217 = 1
				CP-218 = 1 to 10
				CP-2500 = 1 to 14
				CP-2520 = 1 to 15

	7	PARAM	Address input	Head address of set data (MW, DW, #W)
Output	1	BUSY	B-VAL	Message is being received.
	2	COMPLETE	B-VAL	The receiving of the message has been completed.
	3	ERROR	B-VAL	Occurrence of error

* : Indicates the I/O designation at the CP-717.

** : Designate the MEMOBUS protocol (= 1) if transmission is to be performed with the MELSEC or OMRON procedure. Protocol conversion will be carried out at the transmission device (CP-217, CP-218). Refer to (1) of 5.3.4, "OMRON Communication" or (2) of 5.3.4, "MELSEC Communication" of the Control Pack CP-9200SH User's Manual (SIE-C879-40.1) for details on the protocol conversion specifications.

PARAM

No.	IN/OUT	Contents		Remarks
		MEMOBUS	Process result	
00	OUT	Process result	Process result	
01	OUT	Status	Status	
02	OUT*	Calling station #	Calling station #	* Calling connection # in the case of DEV-TYP = CP-218.
03	SYS	System reserved	System reserved	
04	OUT	Function code		
05	OUT	Data address	Data address	
06	OUT	Data size	Data size	
07	OUT	Calling CPU #	Calling CPU #	
08	IN	Coil offset		
09	IN	Input relay offset		
10	IN	Input register offset		
11	IN	Holding register offset		
12	IN	Write-in range LO		
13	IN	Write-in range HI		
14	SYS	For system use	For system use	
15	SYS	System reserved	System reserved	
16	SYS	System reserved	System reserved	

* When CP-218 is set for DEV-TYP, IN.

7.9.1 Parameters

(1) Process Result (PARAM00)

The process result is output to the upper byte. The lower byte is for system analysis.

- 00□□ : In process (BUSY)
- 10□□ : End of process (COMPLETE)
- 8□□□ : Occurrence of error (ERROR)

[Error Classification]

- 81□□ : Function code error
  An unused function code was received.
- 82□□ : Address setting error
  The data address, coil offset, input relay offset, input register offset, or holding regist offset setting is out of range.
- 83□□ : Data size error
  The size of the sent or received data is out of range.
- 84□□ : Line No. setting error
  The line No. setting is out of range.
- 85□□ : Channel No. setting error
  The channel No. setting is out of range.
- 86□□ : Station address error
  The station No. setting is out of range.
- 88□□ : Transmission unit error.
  An error response was returned from the transmission unit. (Refer to (2) of 7.9.1.)
- 89□□ : Device selection error
  A non-applicable device is selected.

(2) Status (PARAM01)

Outputs the status of the transmission unit. See 7.8.1 (2), "Status (PARAM01)" for details.

(3) Calling Station # (PARAM02)

[CP-215, CP-216, CP-217, CP-2500, CP-2520]

The station number of sending side is output.

[CP-218]

1 to 20: The calling station connection number is set.

(4) Function Code (PARAM04)

Outputs the MEMOBUS function code received.

Function code		Output
00H	Unused	×
01H	Read coil status	○
02H	Read input relay status	○
03H	Read contents of holding register	○
04H	Read contents of input register	○
05H	Change status of single coil	○
06H	Write into a single holding register	○
07H	Unused	×
08H	Loop-back test	○
09H	Read contents of holding register (expanded)	○
0AH	Read contents of input register (expanded)	○
0BH	Write into holding register (expanded)	○
0CH	Unused	×
0DH	Discontinuous readout of holding register (expanded)	○
0EH	Discontinuous write into holding register (expanded)	○
0FH	Change status of a multiple coil	○
10H	Write into a plurality of holding registers	○
11H to 20H	Unused	×
21H to 3FH	System reserved	×
40H to 4FH	System reserved	×
50H to	Unused	×

(× : cannot be output, ○ : can be output)

Note : The MB, MW, IB, and IW registers can be used respectively as the coil, holding register, input relay, and input registers during slave operation.

(5) Data Address (PARAM05)

The data address requested by the sending side is output.

(6) Data Size (PARAM06)

The data size (number of bits or number of words) of the requested read or write is output.

(7) Calling CPU # (PARAM07)

The calling CPU No. is output.

When the sending source is CP-9200SH, 1 or 2 is output. For other cases, 0 is output.

(8) Coil Offset (PARAM08)

Set the offset word address of the coil.

This is valid in the case of function codes 01H, 05H, and 0FH.

(9) Input Relay Offset (PARAM09)

Set the offset word address of the input relay.

This is valid in the case of function code 02H.

(10) Input Register Offset (PARAM10)

Set the offset word address of the input register.

This is valid in the case of function codes 04H and 0AH.

(11) Holding Register Offset (PARAM11)

Set the offset word address of the hold register.

This is valid in the case of function codes 03H, 06H, 09H, 0BH, 0DH, 0EH, and 10H.

(12) Write-in Range LO (PARAM12), Write-in Range HI (PARAM13)

Set the write allowable range for the request for write-in. A request which is outside of this rang will cause an error.

This is valid in the case of function code 0BH, 0EH, 0FH, and 10H.

0 ≦ Write-in Range LO ≦ Write-in Range HI ≦ Maximum value of MW Address

(13) For System Use (PARAM14)

The channel No. being used is stored. Make sure that this will be set to 0000H by the use program on the first scan after turning on the power. This value must not be changed by the use program thereafter since this parameter will then be used by the system.

(14) When Non-procedural is set for Transmission Protocol

PARAM04 has no function. The settings of PARAM08, PARAM09, and PARAM10 are no necessary. The message receivable register is only MW.

7.9.2 Inputs

(1) EXECUTE (Receive Message Execution Command)

When this command becomes "ON", the message is received.

This must be held until COMPLETE (completion of process) or ERROR (occurrence of error becomes "ON".

(2) ABORT (Receive Message Forced Interruption Command)

This command forcibly interrupts the receiving of the message. This has priority over EXECUT (receive message execution command).

(3) DEV-TYP (Transmission Device Type)

Designates transmission device type.

	Transmission Device Type
CP-215	1
CP-216	4
CP-217	5
CP-218	6
CP-2500	3
CP-2520	7

(4) PRO-TYP (Transmission Protocol)

Designates transmission protocol. When transmitting with MELSEC or OMRON procedures, designate MEMOBUS protocol (=1). Protocol is converted by the transmission device (CP-217, CP-218).

MEMOBUS: Setting = 1
Non-procedural: Setting = 2

For details of protocol conversion specifications, refer to the following manuals.

Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (1) "OMRON communications"
5.3.4 (2) "MELSEC communications"

Note: In non-procedural transmission, a response is not sent to the other station.

(5) CIR-NO (Line No.)

Designate the Circuit No.

	Circuit No.
CP-215	1 to 8 (Option)
CP-216	1 to 8 (Option)
CP-217	1 to 24 (Option)
CP-218	1 to 8 (Option)
CP-2500	1 to 8 (Option)
CP-2520	1 to 8 (Option)

(6) CH-NO (Channel No.)

Designate the channel No. of the transmission unit. However, the channel number should be set so as not to be duplicated on a single line.

	Channel No.
CP-215	1 to 13
CP-216	1 to 8
CP-217	1
CP-218	1 to 10
CP-2500	1 to 14
CP-2520	1 to 15

(7) PARAM (Setting Data Head Address)

The head address of the set data is designated. For details of the setting data, refer to 7.9.1. "Parameters."

9.3 Outputs

(1) Busy (In Process)

Indicates that the process is being executed. Keep EXECUTE set to "ON".

(2) COMPLETE (Completion of Process)

Becomes "ON" for only 1 scan upon normal completion.

(3) ERROR (Occurrence of Error)

Becomes "ON" for only 1 scan upon occurrence of error.
Refer to PARAM00 (7.8.1 (1)) and PARAM01 (7.8.1 (2)) concerning the cause.

7.9.4 Limitations Arising from Other Companies' Communications Protocols with the CP-217IF

(1). When Making a Dedicated Protocol Connection Link with the MELSEC Computer

■ Communication is possible with type 1 protocol (response possible only for full-dual connection

■ With a MSG-RCV function, receiving and sending with response of ACPU common comman to and from the MELSEC master device are possible, but commands that may be used a limited (read out/write in of device memory, wrap test).

■ Designate MEMOBUS protocol (= 1) by input of the PRO-TYP (transmission protocol) of th MSG-RCV function. On the I/O definition screen for the transmission port, if MELSEC slave set, conversion to the corresponding MELSEC format is performed by the CP-217IF unit. Change designated parameters to parameters of corresponding MEMOBUS procedures. Ref to the following manuals for correspondence of MELSEC commands and MEMOBUS functi codes, correspondence of registers for sending and receiving and device addresses on th MELSEC side.
· Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (2) "MELSEC communications"

(2) When Making an OMRON Upward Linking Mode (SYSWAY) Connection

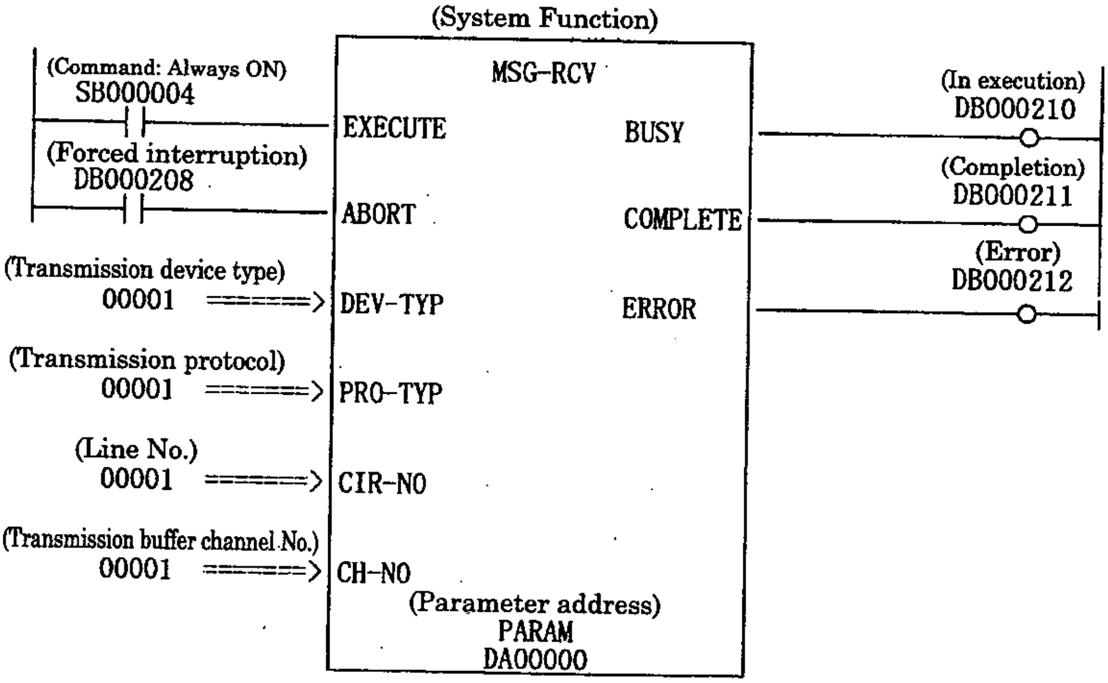
■ With a MSG-RCV function, receiving and sending with responce of commands to and from th OMRON master device are possible, but commands that may be used are limited (I/O rela DM read out/write, wrap test).

■ Designate MEMOBUS protocol (= 1) for input of the PRO-TYP (transmission protocol) of th MSG-RCV function. On the I/O definition screen for the transmission port, if OMRON slave set, conversion to the corresponding OMRON format is performed by the CP-217IF unit. Change designated parameters to parameters of corresponding MEMOBUS procedures. Refer to the following manuals for correspondence of OMRON commands and MEMOBU function codes, regarding correspondence of registers for sending and receiving and the rela (CH)/DM area on the OMRON side.
· Control Pack CP-9200SH User's Manual (SIE-C879-40.1)
5.3.4 (1) "OMRON communications"

■ This corresponds to transmission procedures by multi-programs stipulated in OMRON proc dures, but set the upper limit for the number of words that can be accessed with one instru tion to 125 words for DM register read out, and 100 words for writing (restricted conditions MEMOBUS procedures).

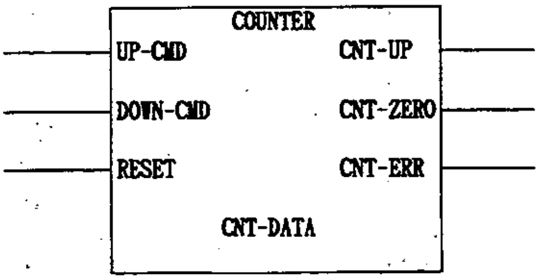
9.5 Program Example

```
(Set the system register to 0 on the first scan.)
|  SB000003
├───┤ ├──

 [├ 00000]                                        [⇒ DW00014]

                                                  (Write-in range LO)
  ├ 0                                              ⇒ DW00012

                                                  (Write-in range HI)
  ├ 32767                                          ⇒ DW00013
```



```
|  DB000211
├───┤ ├──

 (Pass counter)
 [├ INC   DW00024]
|  DB000212
├───┤ ├──

IFON

(Error counter)
INC DW00025

(Store process result)
  ├ DW00000                                       ⇒ DW00026

  (LINK status)
  ├ DW00001                                       ⇒ DW00027

IEND

DEND
```

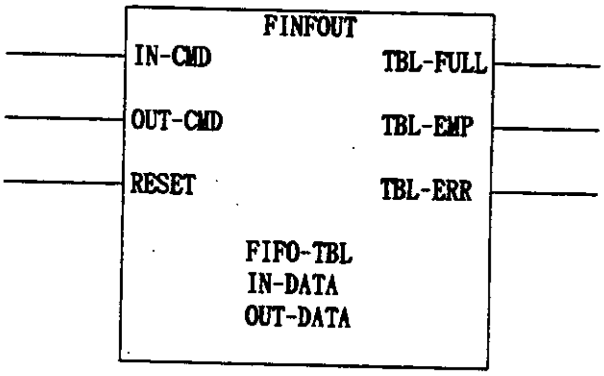
7.10 Counter Function (COUNTER)

Name of Function	COUNTER
Function	Increments or decrements the current value when the count up/down command (UP-CMD, DOWN-CMD) changes from OFF to ON. When the counter reset command (RESET) becomes ON, the current counter value is set to 0. Also, the current counter value and the set value are compared and the comparison result is output. * The current value will not be incremented neither decremented if a counter error (current value > set value) occurs.
Function Definition	 COUNTER UP-CMD DOWN-CMD RESET CNT-UP CNT-ZERO CNT-ERR CNT-DATA

I/O Definition	No.	Name	I/O Designation*	Description	
Input	1	UP-CMD	B-VAL	Count up command (OFF → ON)	Data area for counter process 1: Set value 2: Current value 3: Work flag
	2	DOWN-CMD	B-VAL	Count down command (OFF → ON)	
	3	RESET	B-VAL	Counter reset command	
	4	CNT-DATA	Address input	Head address of data area for counter process (MW or DW register)	
Output	1	CNT-UP	B-VAL	Becomes ON when current counter value = set value.	
	2	CNT-ZERO	B-VAL	Becomes ON when current counter value = 0.	
	3	CNT-ERR	B-VAL	Becomes ON when current counter value > set value.	

*: Indicates the I/O designation at the CP-717.

11 First-in First-out Function (FINFOUT)

Name of Function	FINFOUT				
Function	This is a first-in first-out type block data transfer function. The FIFO data table is composed of a 4-word header part and a data buffer. 3 words of the header part (data size, input size, output size) must be set before this function is referenced. · When the data input command (IN-CMD) becomes ON, the designated number of data is sequentially stored from the designated input data area to the data area of the FIFO table. · When the data output command (OUT-CMD) becomes ON, the designated number of data are transferred from the head of the data area of the FIFO table to the designated output data area. · When the reset command (RESET) becomes ON, the number (amount) of data stored is set to zero and the FIFO table empty output (TBL-EMP) becomes ON. · If "size of available space for data (empty size) < input size" or if "data size < output size," the FIFO table error (TBL-ERR) becomes ON.				
Function Definition	 FINFOUT IN-CMD OUT-CMD RESET TBL-FULL TBL-EMP TBL-ERR FIFO-TBL IN-DATA OUT-DATA				
I/O Definition	No.	Name	I/O Designation*	Description	
Input	1	IN-CMD	B-VAL	Data input command (IN-CMD)	FIFO Table Configuration 0 : data size 1 : input size 2 : output size 3 : number of data stored 4 : data
	2	OUT-CMD	B-VAL	Data output command (OUT-CMD)	
	3	RESET	B-VAL	Reset command	
	4	FIFO-TBL	Address input	Head address of FIFO table (MW or DW address)	
	5	IN-DATA	Address input	Head address of input data (MW or DW address)	
	6	OUT-DATA	Address input	Head address of output data (MW or DW address)	
Output	1	TBL-FULL	B-VAL	FIFO table is full.	
	2	TBL-EMP	B-VAL	FIFO table is empty.	
	3	TBL-ERR	B-VAL	FIFO table error	

* : Indicates the I/O Designation at the CP-717.

APPENDIX

The contents of Appendix are as follows:

Appendix A: Types of Instruction Words
Appendix B: List of Instructions
Appendix C: Differences on Programming between CP-9200H and CP-9200SH

The data type (bit type, integer type, double-length integer type, real number type) that can be used will differ for each instruction. Refer to Chapter 4 "BASIC INSTRUCTIONS" for details.

A Types of Instruction Words

Type of Instruction Word	Instruction Words
Program control instruction	SEE FOR WHILE ON/OFF IFON/IFOFF ELSE END FSTART FIN FOUT COMMENT XCALL
Direct I/O instruction	INS OUTS
Relay circuit instruction	─┤├─ ─┤/├─ ─┤↑├─ ─┤↓├─ ─┤ᵀ ├─ ─┤ ᵀ├─ ─┤ˢ ├─ ─┤ ˢ├─ ─○─┤ ─[S]─┤ ─[R]─┤ ─↓─ ─↑─ ─⤴
Logical operation instruction	∧ ∨ ⊕
Numerical operation instruction	├ ‖├ ⟹ + − ++ -- × ÷ INC DEC MOD REM TMADD TMSUB SPEND
Numerical conversion instruction	INV COM ABS BIN BCD PARITY ASCII BINASC ASCBIN
Numerical comparison instruction	< ≦ = ≠ ≧ > RCHK
Data operation instruction	ROTL ROTR MOVB MOVW XCHG SETW BEXTD BPRESS BSRCH SORT SHFTL SHFTR COPYW BSWAP
Basic function instruction	SQRT SIN COS TAN ASIN ACOS ATAN EXP LN LOG
DDC instruction	DZA DZB LIMIT PI PD PID LAG LLAG FGN IFGN LAU SLAU PWM
Table data operation instruction	TBLBR TBLBW TBLSRL TBLSRC TBLCL TBLMV QTBLR QTBLRI QTBLW QTBLWI QTBLCL
SFC instruction	SFC ╪ Ƶ + ABOX SBOX AEND SFCSTEP
System function	COUNTER FINFOUT TRACE DTRC-RD FTRC-RD ITRC-RD MSG-SND MSG-RCV ISET-213 ICNS-WR ICNS-RD

List of Instructions

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
ogram ntrol structions	SEE child drawing	SEE	○	Designate the No. of the child or grandchild drawing to be referenced after "SEE" SEE H01	○	○
	FOR statement	FOR FEND		Loop execution statement - 1 FOR V = a to b by c V : arbitrary integer register May specify as I or J. a, b, c : May specify an arbitrary integer. (b > a > 0, c > 0) FEND: END of FOR instruction	○	○
	WHILE statement	WHILE ON/OFF WEND		Loop execution statement - 2 WEND : END of WHILE-ON OFF instruction	○	○
	IF statement	IFON/IFOFF ELSE IEND		Conditional execution statement IEND: END of IFON/IFOFF instruction	○	○
	END	FEND WEND IEND DEND		The exclusive END instruction is indicated automatically by the CP-717 for each of the above statements. DEND is indicated for the END of a drawing. Only "END" is accepted as an input from the CP-717; FEND, WEND, etc. will not be accepted.	○	○
	COMMENT	"nnnnnnnn"		Character strings enclosed in " " will be handled as a comment.	○	○

Note) ○ mark in the "[ ] Instruction" column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-920
Program control instructions	Function I/F	FSTART		Function referencing instruction	○	○
		FIN		Function input instruction Stores input data from the designated input register into the function input register. Designated input register B-VAL : CPU internal register (B register) I-VAL : CPU internal register (A register) L-VAL : CPU internal register (A register) F-VAL : CPU internal register (F register) I-REG : arbitrary integer register L-REG : arbitrary double-length integer register F-REG : arbitrary real number register Address input	○	○
		FOUT		Function output instruction Stores output data from the function output register to the designated output register. Designated output register B-VAL : CPU internal register (B register) I-VAL : CPU internal register (A register) L-VAL : CPU internal register (A register) F-VAL : CPU internal register (F register) I-REG : arbitrary integer register L-REG : arbitrary double-length integer register F-REG : arbitrary real number register	○	○
	Expansion program execution instruction	XCALL	○	Instruction for referencing an expansion program[1].	○	
Direct I/O Instructions	Input instruction (interruption prohibited)	INS	○	INS MA00100 ———○—┤ Data input and storage are executed with interruption prohibited.	○	
	Output instruction (interruption prohibited)	OUTS	○	OUTS MA00100 ———○—┤ The setting and output of data are executed with interruption prohibited.	○	

[1]: There are four types of expansion programs which reference this instruction: constant table (M register I/O conversion table, interlock table, and part composition table.

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the val of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
elay ircuit nstructions	NO contact	┤├		No limit in the serial circuit. Bit type designation of any register as a relay number is possible (MB00011A).	○	○
	NC contact	┤/├		No limit in the serial circuit. Bit type designation of any register as a relay number is possible (MB00011A).	○	○
	Rise pulse	┤↑├		No limit in the serial circuit. Bit type designation of any register as a relay number is possible (MB00011A).	○	○
	Fall pulse	┤↓├		No limit in the serial circuit. Bit type designation of any register as a relay number is possible (MB00011A).	○	○
	On-delay timer (Unit of measurement: 10 ms)	┤[T]├		Set value: count register ┤[T ├ Set value: any register, constant (setting unit: 10ms) Count register : M or D register	○	○
	Off-delay timer (Unit of measurement: 10 ms)	┤[T]├			○	○
	On-delay timer (Unit of measurement: 1s)	┤[S]├		Set value: count register ┤[S ├ Set value: any register, constant (setting unit: 1s) Count register : M or D register	○	
	Off-delay timer (Unit of measurement: 1s)	┤[S]├			○	
	Coil	─○┤		MB000000 ├ MW00200 = 00001 ───○─┤ MB000000 ├───┤├─── IFON	○	○
	Set coil	┤S├		MB000000 MB000010 ├───┤├───────{S}─┤ By turning MB000000 "ON," MB000010 turns "ON." Subsequently, even if MB000000 turns "OFF," it stays "ON."	○	○
	Reset coil	┤R├		MB000020 MB000010 ├───┤├───────{R}─┤ By turning MB000020 "ON," MB000010 turns "OFF." Subsequently, even if MB000020 turns "OFF," it stays "OFF."	○	○
	Branching/ convergence point instruction	─┬──┬──┘		A branching or converging indication can be attached to any of the above relay type instructions.	○	○

Note) ○ mark in the "[ ] Instruction" column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-920
Logical Operation Instructions	AND	∧	○	Integer type designation of any register or constant is possible.	○	○
	OR	∨	○	Integer type designation of any register or constant is possible.	○	○
	Exclusive OR	⊕	○	Integer type designation of any register or constant is possible.	○	○
Numerical Operation Instructions	Integer type entry	⊢	○	Starts integer type operation. ⊢ MW00280+00100 ⇒MW00220	○	○
	Real number type entry	⊩	○	Starts real number type operation. ⊩ MW00280+00100 ⇒MW00220	○	○
	Store	⇒	○	Stores operation result in designated register.	○	○
	Add	+	○	Ordinary numerical addition (with operation error).* ⊢ MW00280+00100 ⇒MW00220 All registers and constants can be designated	○	○
	Subtract	—	○	Ordinary numerical subtraction (with operation error).* ⊢ MW00280-00100 ⇒MW00220 All registers and constants can be designated.	○	○
	Extended add	++	○	Closed numerical addition (without operation error). 32768+1=-32768 0 → 32767 → -32768 → 0	○	○
	Extended subtract	--	○	Closed numerical subtraction (without operation error). -32768-1=32767 0 → -32768 → 32767 → 0	○	○
	Multiply	×	○	In the case of integer type and double-length integer type, use × and ÷ in combination.	○	○
	Divide	÷	○		○	○

*: On the CP-9200H, an operation error will not occur with double-length operations. On the CP-9200S operation error will occur with double-length operations.

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the val of the immediately preceding B register) can be added to the instruction .

continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
umerical peration istructions	Increment	INC	○	Adds 1 to the designated register. INC MW00100 If MW00100= 99, the operation result = 100.	○	○
	Decrement	DEC	○	Subtracts 1 from the designated register. DEC MW00100 If MW00100= 99, the operation result = 98.	○	○
	Integer type remainder	MOD	○	⊢ MW00100×01000÷00121 MOD ⇒MW00101 In this example, the remainder of division is taken out.	○	○
	Real number type remainder	REM	○	‖⊢ MF00200 REM 1.5 ⇒MF00202 In this example, the remainder of division is taken out.	○	○
	Time addition	TMADD	○	Addition of hrs/min/sec TMADD MW00000, MW00100	○	
	Time subtraction	TMSUB	○	Subtraction of hrs/min/sec TMSUB MW00000, MW00100	○	
	Time spend	SPEND	○	Finds elapsed time between two times. (Difference in yr/mo/day/hr/min/sec in total number of seconds.) SPEND MW00000, MW00100	○	

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-920
Numerical Conversion Instructions	Sign inversion	INV	○	⊢MW00100 INV If MW00100= 99, the operation result = -99.	○	○
	Complement of 1	COM	○	⊢MW00100 COM If MW00100=FFFFH, the operation result=0000H	○	○
	Absolute value conversion	ABS	○	⊢MW00100 ABS If MW00100=-99, the operation result=99	○	○
	Binary conversion	BIN	○	⊢MW00100 BIN If MW00100=1234H (hexadecimal), the operation result = 01234 (decimal).	○	○
	BCD conversion	BCD	○	⊢MW00100 BCD If MW00100= 1234 (decimal), the operation result = 1234H (hexadecimal).	○	○
	Parity conversion	PARITY	○	Calculates the number of binary expression bits that are ON (= 1). ⊢MW00100 PARITY If MW00100= F0F0H, the operation result = 8.	○	○
	ASCII conversion 1	ASCII	○	The designated character string is converted to ASCII code and substituted in the register. ASCII MW00200 "ABCDEFG"	○	
	ASCII conversion 2	BINASC	○	Sixteen-bit binary data is converted to four-digit hexadecimal ASCII code. BINASC MW00100	○	
	ASCII conversion 3	ASCBIN	○	The numerical value indicated by a four-digit hexadecimal ASCII code is converted to 16-bit binary data. ASCBIN MW00100	○	

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the valu of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
umerical omparison nstructions	<	<	○	ON or OFF is left in the B register as a result of the comparison instruction. MB000010 ├ MW00000<10000 ——○—┤ │ MB000010 ├——┤├—— IFON	○	○
	≦	≦	○		○	○
	=	=	○		○	○
	≠	≠	○		○	○
	≧	≧	○		○	○
	>	>	○		○	○
	Range check	RCHK	○	Checks whether the value in the A register is in range or not. Lower limit Upper limit ├ MW00100 RCHK -1000, 1000 If it is in range B register turns ON, if out of range, OFF.	○	
ata peration nstructions	Bit rotation (L) (left rotation)	ROTL	○	Bit-addr Count Width ROTL MB00100A → N = 1 W = 20	○	
	Bit rotation (R) (right rotation)	ROTR	○	Bit-addr Count Width ROTR MB00100A → N = 1 W = 20	○	
	Bit transfer	MOVB	○	Source Desti. Width MOVB MB00100A →MB00200A W = 20	○	
	Word transfer	MOVW	○	Source Desti. Width MOVW MW00100 → MW00200 W = 20	○	○
	Exchange	XCHG	○	Source1 Source2 Width XCHG MW00100 → MW00200 W = 20	○	○
	Table initialization	SETW	○	Desti. Data. Width SETW MW00200 → D = 00000 W = 20	○	
	Byte → word development	BEXTD	○	The binary data string stored in the word form register area is developed a byte at a time into words. BEXTD MW00100 to MW00200 B = 10	○	
	Word → byte compression	BPRESS	○	The lower byte only of the word data stored in the word form register area are gathered into a byte string. BPRESS MW00100 to MW00200 B = 10	○	

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-920
Data Operation Instructions	Data search	BSRCH	○	A search is made, within the designated register range, for the position of data which match the stipulated data. BSRCH MW00000 W = 20 D = 100 R = MW00100	○	
	Sort	SORT	○	A sort is performed on registers within the designated register range. SORT MW00000 W = 100	○	
	Bit shift left	SHFTL	○	The designated bit strings are shifted to the left. SHFTL MB00100A N = 1 W = 20	○	
	Bit shift right	SHFTR	○	The designated bit strings are shifted to the right. SHFTR MB00100A N = 1 W = 20	○	
	Word copy	COPYW	○	The designated register range is copied. Even if there is overlap between the copy destination and copy source, the copy will be correctly performed. COPYW MW00100 → MW00200 W = 20	○ ○	
	Byte swap	BSWAP	○	The upper and lower bytes of the designated word variable are swapped. BSWAP MW00100		

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the val of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
asic* unction nstructions	Square root	SQRT	○	Taking the square root of a negative number will result in the square root of the absolute value multiplied by -1. ⊩MF00100 SQRT	○	○
	Sine	SIN	○	Input = in degrees ⊩MF00100 SIN	○	○
	Cosine	COS	○	Input = in degrees ⊩MF00100 SIN	○	○
	Tangent	TAN	○	Input = in degrees ⊩MF00100 TAN	○	○
	Arc sine	ASIN	○	⊩MF00100 ASIN	○	○
	Arc cosine	ACOS	○	⊩MF00100 ACOS	○	○
	Arc tangent	ATAN	○	⊩MF00100 ATAN	○	○
	Exponent	EXP	○	⊩MF00100 EXP $e^{MF00100}$	○	○
	Natural log	LN	○	⊩MF00100 LN $\log_{e}(MF00100)$	○	○
	Common log	LOG	○	⊩MF00100 LOG $\log_{10}(MF00100)$	○	○

When using a basic function instruction with integer type data, scaling is necessary. For details, refer to Chapter 4 "BASIC INSTRUCTIONS".

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-920
DDC Instructions	Dead zone A	DZA	○	⊢MW00100 DZA 00100	○	○
	Dead zone B	DZB	○	⊢MW00100 DZB 00100	○	○
	Upper/lower limit	LIMIT	○	⊢MW00100 LIMIT -00100 00100	○	○
	PI control	PI	○	⊢MW00100 PI MA00 200	○	○
	PD control	PD	○	⊢MW00100 PD MA00200	○	○
	PID control	PID	○	⊢MW00100 PID MA00200	○	○
	First-order lag	LAG	○	⊢MW00100 LAG MA00200	○	○
	Phase-lead-lag	LLAG	○	⊢MW00100 LLAG MA00200	○	○
	Function generator	FGN	○	⊢MW00100 FGN MA00200	○	○
	Inverse function generator	IFGN	○	⊢MW00100 IFGN MA00200	○	○
	Linear accelerator unit 1	LAU	○	⊢MW00100 LAU MA00200	○	○
	Linear accelerator unit 2	SLAU	○	⊢MW00100 SLAU MA00200	○	○
	Pulse width modulation	PWM	○	⊢MW00100 PWM MA00200	○	○

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the valu
of the immediately preceding B register) can be added to the instruction .

ontinued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
ble Data peration structions	Block read	TBLBR	○	TBLBR TBL1, MA00000, MA00100	○	
	Block write	TBLBW	○	TBLBW TBL1, MA00000, MA00100	○	
	Row search (vertical)	TBLSRL	○	TBLSRL TBL1, MA00000, MA00100	○	
	Column search (horizontal)	TBLSRC	○	TBLSRC TBL1, MA00000, MA00100	○	
	Block transfer between tables	TBLMV	○	TBLMV TBL1, TBL2, MA00000	○	
	Cue table read (pointer stationary)	QTBLR	○	QTBLR TBL1, MA00000, MA00100	○	
	Cue table read (pointer advances)	QTBLRI	○	QTBLRI TBL1, MA00000, MA00100	○	
	Cue table write (pointer stationary)	QTBLW	○	QTBLW TBL1, MA00000, MA00100	○	
	Cue table write (pointer advances)	QTBLWI	○	QTBLWI TBL1, MA00000, MA00100	○	
	Clear cue pointer	QTBLCL	○	QTBLCL TBL1	○	

Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200
SFC Instructions	SFC execution	SFC		SFC EXECUTE OUT MA□□□□□	○	○
	NO contact transition judgment	╪		Designation of transition condition ╪ IB0010A (Cannot modify with a subscript.)	○	○
	NC contact transition judgment	≠		Designation of transition condition ≠ MB0012B (Cannot modify with a subscript.)	○	○
	Timer transition judgment	+		Transition timer set value + 10.00 (Cannot modify with a subscript.)	○	○
	Action box	ABOX		ABOX S10: The corresponding program is executed on each scan after transition to step box S10 and until transition to the next step.	○	○
		SBOX		SBOX S11: The corresponding program is executed just once upon transition to step box S11.	○	○
	End action box	AEND		End of SFC action box.	○	○
	Branching/ convergence point instruction	├ ├ ┘		Designation of branching point, convergence point, and convergence connection of SFC.	○	○
	SFC step entry	SFCSTEP	○	SFCSTEP STEP name → DW00000 Store system STEP No. of designated STEP in the A register.	○	

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the valu of the immediately preceding B register) can be added to the instruction .

(continued)

Type	Name	Symbol	[ ] Instruction	Description	Device Model	
					CP-9200SH	CP-9200H
ystem unctions	Counter	COUNTER		Up/down counter	○	○
	First-in first-out	FINFOUT		First-in first-out function	○	○
	Trace function	TRACE*1		Write-in of trace data into the data trace memory.	○	○
	Data trace read function	DTRC-RD*2		Readout of data from data trace memory to user memory	○	○
	Failure trace read function	FTRC-RD		Readout of data from failure trace memory to user memory.	○	
	Inverter trace read function	ITRC-RD		Readout of data from inverter trace memory to user memory.	○	
	Send message function	MSG-SND*1		Send CP-215/CP-216/CP-217/CP-218/ CP-2500 message.	○	○
	Receive message function	MSG-RCV*1		Receive CP-215/CP-216/CP-217/CP-218/CP2500 message.	○	○
	Inverter constant write function	ICNS-WR		Applies to the inverter connected to CP-215 or CP-216.	○	
	Inverter constant read function	ICNS-RD		Applies to the inverter connected to CP-215 or CP-216.	○	
	CP-213 initial data setting	ISET-213		Sets the initial data for the inverter connected to the CP-213 line.	○	

: The CP-9200SH and the CP-9200H are slightly different.

: Equivalent to TRACE-RD function on the CP-9200H.

(Note) ○ mark in the "[ ] Instruction " column means that "[ ]" (conditional execution according to the value of the immediately preceding B register) can be added to the instruction .

C Differences on Programming between CP-9200H and CP-9200SH

For details of each instruction, refer to Chapter 4 "BASIC INSTRUCTIONS".

		Model	CP-9200SH		CP-9200H	Remarks
Item			1MB	2MB		
1	Additional instructions	Program control instruction	XCALL		None	
		Data transfer instruction	ROTR, ROTL, MOVB, SETW, COPYW, SHL, SHR			
		DDC instruction	RCHK			
		SFC instruction	SFCSTEP			
		System function	FTRC-RD			
		Sequence instruction	⊣S⊢ (set coil) ⊣R⊢ (reset coil)			
2	Modified instructions	DDC instruction	LAU (with both functions of LAU and VLAU)		LAU and VLAU	
			SLAU (with both functions of SLAU and VSLAU)		SLAU and VSLAU	
		System function	DTRC-RD		TRACE-RD	
			TRACE		TRACE	
			MSG-SND		SND	
			MSG-RCV		RCV	
		Direct I/O instruction	INS, OUTS		IN, OUT	
3	Deleted instructions	DDC instruction	None		LPID	· As CP-9200SH has no memory card connection function, the functions related to memory card (MC-WRITE, MC-READ, MC-CHK) are deleted. · CP-9200SH supports double-length integer multiplication/division function (LMUL, LDIV) by the instructions × and ÷ .
		System function	None		MC-WRITE MC-REA MC-CHK	
			None		LMUL	
			None		LDIV	
4	Application capacity		equivalent to 12 k steps/CPU		equivalent to 4 k steps/CPU	
5	Data memory	Register common to all DWGs (M)	32 k words/CPU		16128 words (common for CPUs)	· With CP-9200H, M, I, and registers are common for CPU0 and CPU1. With CP-9200SH, they are unique each for CPU1 and CPU2. · With CP-9200H, D register is common to all DWG's. With CP-9200SH, it is unique to each DWG. · With CP-9200H, I and O registers are cleared at the power turned ON. With CP 9200SH, they are not cleared at the power turnec ON. · The number and contents o S register are different between CP-9200H and CP 9200SH.
		Input register (I)	5 k words/CPU		128 words (common for CPUs)	
		Output register (O)	5 k words/CPU		128 words (common for CPUs)	
		System register (S)	1 k words/CPU		256 words/CPU	
		Register unique to each DWG (D)	Max. 16 k words/DWG, function		2 k words/CPU	
		DWG constant register (#)	Max. 16 k words/DWG, function		Max. 512 words/DWG	
		Common constant register (C)	16 k words/CPU		None	

continued)

		Model	CP-9200SH		CP-9200H	Remarks
tem			1MB	2MB		
6	Trace memory	Data trace	Max. 128 k words (32 k words × 4 groups)/CPU		192 k words (common for CPUs) (32 k words × 3 groups)	· With CP-9200SH, when the trace memory is not used, it can be used for user program area.
		Failure trace	Max. 4 k words (64 items × 450)/CPU		None	
7	Table programming		Possible		Not possible	
8	Drawing/ function capacity	Starting (A)	64 drawings		32 drawings	
		High-speed scan (H)	100 drawings		32 drawings	
		Low-speed scan (L)	100 drawings		32 drawings	
		Interruption (I)	64 drawings		32 drawings	
		User function	100 functions		32 drawings	
		Number of steps/DWG, function	500 steps		300 steps	
		Drawing hierarchy	3 lays		2 lays	
9	Shared memory between CPUs		Possible when M register is set on the screen		M register	
0	Program secret protection		Possible in units of drawing		Possible in units of CPU	
1	Calendar function		Provided		Not provided	
2	MEMBUS I/F		M and I register (possible by CPU)		S, I, O, M, and D register	
3	Servo parameter	Area	Fixed I/O register (128 words/axis) (IWC000 to IWFFFF, OWC000 to OWFFFF)		Common with M register (50 words/axis) (MW00000 to MW00399)	· For CP-9200SH, the number and arrangement of servo parameters and their functions are partly different from those of CP-9200H.
		Servo fixed parameter	Settings on the screen (separated from servo parameter)		Setting of M register (included in servo parameter)	
4	Temperature input		The system function MSG-SND is used.		Temperature input display	
5	Compatibility of user program		Provided with source conversion tool to convert the user program for CP-9200H to that for CP-9200SH.		—	
16	Batch loading		At batch loading, program memory and data memory (S, I, O, M, and D register) for each CPU are cleared.		At batch loading, program memory and data memory (S and D register) for each CPU are cleared, but M register is not cleared.	

MACHINE CONTROLLER CP-9200SH
PROGRAMMING MANUAL

TOKYO OFFICE
New Pier Takeshiba South Tower, 1-16-1, Kaigan, Minatoku, Tokyo 105-6891 Japan
Phone 81-3-5402-4511 Fax 81-3-5402-4580

YASKAWA ELECTRIC AMERICA, INC.
2121 Norman Drive South, Waukegan, IL 60085, U.S.A.
Phone 1-847-887-7000 Fax 1-847-887-7370

MOTOMAN INC. HEADQUARTERS
805 Liberty Lane West Carrollton, OH 45449, U.S.A.
Phone 1-937-847-6200 Fax 1-937-847-6277

YASKAWA ELÉTRICO DO BRASIL COMÉRCIO LTDA.
Avenida Fagundes Filho, 620 Bairro Saude-Sao Päulo-SP, Brazil CEP: 04304-000
Phone 55-11-5071-2552 Fax 55-11-5581-8795

YASKAWA ELECTRIC EUROPE GmbH
Am Kronberger Hang 2, 65824 Schwalbach, Germany
Phone 49-6196-569-300 Fax 49-6196-888-301

Motoman Robotics Europe AB
Box 504 S38525 Torsås, Sweden
Phone 46-486-48800 Fax 46-486-41410

Motoman Robotec GmbH
Kammerfeldstraβe 1, 85391 Allershausen, Germany
Phone 49-8166-900 Fax 49-8166-9039

YASKAWA ELECTRIC UK LTD.
1 Hunt Hill Orchardton Woods Cumbernauld, G68 9LF, United Kingdom
Phone 44-1236-735000 Fax 44-1236-458182

YASKAWA ELECTRIC KOREA CORPORATION
Kfpa Bldg #1201, 35-4 Youido-dong, Yeongdungpo-Ku, Seoul 150-010, Korea
Phone 82-2-784-7844 Fax 82-2-784-8495

YASKAWA ELECTRIC (SINGAPORE) PTE. LTD.
151 Lorong Chuan, #04-01, New Tech Park Singapore 556741, Singapore
Phone 65-282-3003 Fax 65-289-3003

YASKAWA ELECTRIC (SHANGHAI) CO., LTD.
4F No.18 Aona Road, Waigaoqiao Free Trade Zone, Pudong New Area, Shanghai 200131, China
Phone 86-21-5866-3470 Fax 86-21-5866-3869

YATEC ENGINEERING CORPORATION
Shen Hsiang Tang Sung Chiang Building 10F 146 Sung Chiang Road, Taipei, Taiwan
Phone 886-2-2563-0010 Fax 886-2-2567-4677

YASKAWA ELECTRIC (HK) COMPANY LIMITED
Rm. 2909-10, Hong Kong Plaza, 186-191 Connaught Road West, Hong Kong
Phone 852-2803-2385 Fax 852-2547-5773

BEIJING OFFICE
Room No. 301 Office Building of Beijing International Club, 21
Jianguomenwai Avenue, Beijing 100020, China
Phone 86-10-6532-1850 Fax 86-10-6532-1851

TAIPEI OFFICE
Shen Hsiang Tang Sung Chiang Building 10F 146 Sung Chiang Road, Taipei, Taiwan
Phone 886-2-2563-0010 Fax 886-2-2567-4677

SHANGHAI YASKAWA-TONGJI M & E CO., LTD.
27 Hui He Road Shanghai China 200437
Phone 86-21-6531-4242 Fax 86-21-6553-6060

BEIJING YASKAWA BEIKE AUTOMATION ENGINEERING CO., LTD.
30 Xue Yuan Road, Haidian, Beijing P.R. China Post Code: 100083
Phone 86-10-6233-2782 Fax 86-10-6232-1536

SHOUGANG MOTOMAN ROBOT CO., LTD.
7, Yongchang-North Street, Beijing Economic Technological Investment & Development Area,
Beijing 100076, P.R. China
Phone 86-10-6788-0551 Fax 86-10-6788-2878

YASKAWA ELECTRIC CORPORATION

YASKAWA

Specifications are subject to change without notice
for ongoing product modifications and improvements.

MANUAL NO. SIE-C879-40.3B